Logique

méthodes formelles pour l'étude des programmes

Paul Gochet
Pascal Gribomont

HERMES

Logique

Langue, Raisonnement, Calcul

Collection dirigée par Mario Borillo et Frédéric Nef

La collection Langue, Raisonnement, Calcul *rassemble des ouvrages consacrés à l'étude des langages et des raisonnements, qu'ils soient naturels ou formels, ou de programmation. Au carrefour des sciences cognitives, de la linguistique, de l'informatique, de la théorie formelle des activités symboliques (dialogue, argumentation, apprentissage) cette collection a pour ambition de contribuer à faire émerger une nouvelle problématique scientifique.*

Déjà parus

Jean-Marie PIERREL, *Dialogue oral homme-machine*, 1987.

Jacques JAYEZ, *L'inférence en langue naturelle*, 1988.

Gérard SABAH, *L'intelligence artificielle et le langage*, vol. 1 : *Représentation des connaissances*, 1988 ; vol. 2 : *Processus de compréhension*, 1989.

Frédéric NEF, *Logique et Langage, Essais de logique intensionnelle*, 1988.

Stanislaw LESNIEWSKI, *Sur les fondements de la mathématique*, trad. G. KALINOWSKI, 1989.

Jacques MOESCHLER, *Modélisation du dialogue. Représentation de l'inférence argumentative*, 1989.

Philip MILLER et Thérèse TORRIS, éds., *Formalismes syntaxiques pour le traitement automatique du langage naturel*, 1990.

M. LOTHAIRE, éd., *Mots*, Mélanges offerts à M.P. SCHUTZENBERGER, 1990.

Jacques PITRAT, *Métaconnaissance, futur de l'intelligence artificielle*, 1990.

Paul GOCHET, Pascal GRIBOMONT, *Logique*, méthodes pour l'informatique fondamentale, vol. 1, 1990, réimp. rev. 1991.

Jean-Pierre DESCLÉS, *Langages applicatifs, langues naturelles et cognition*, 1990.

Eric GRÉGOIRE, *Logiques non monotones et intelligence artificielle*, 1990.

Jean-François NOGIER, *Génération automatique de langage et graphes conceptuels*, 1991.

Archibald MICHIELS, *Traitement du langage naturel et Prolog*, 1991.

Stéphane BOUCHERON, *Théorie de l'apprentissage — de l'approche formelle aux enjeux cognitifs —*, 1992.

Eric BILANGE, *Dialogue personne-machine, modélisation et réalisation informatique*, 1992.

Bruno BACHIMONT, *Le contrôle dans les systèmes à base de connaissances — contribution à l'épistémologie de l'intelligence artificielle*, 1992.

Emmanuel SAINT-JAMES, *La programmation applicative (de LISP à la machine en passant par le lambda-calcul)*, 1993.

Paul Gochet
Pascal Gribomont

Logique

Volume 2

méthodes formelles pour l'étude des programmes

HERMES

© Hermès, Paris, 1994

Editions Hermès
14, rue Lantiez
75017 Paris

ISBN 2-86601-395-6
ISSN 0988-0569

Les auteurs remercient la Fondation Universitaire de Belgique et la Fondation Francqui pour leur appui. Ils expriment leur reconnaissance à M. Baudinet, M. Crabbé, Ph. Delsarte, E. Gillet, G. Hansoul, G. Hulin, A. Nissen, A. Petry, J. Pieters, D. Ribbens, J. Riche, Ch. Ronse, A. Thayse, P. Yans, et à tous ceux qui ont relu tout ou partie du manuscrit et dont les remarques ont constitué une aide précieuse.

Les chapitres 2, 4 et 5 ont pour origine une série de séminaires donnés par le second auteur au Centre de Recherche en Informatique de Nancy. Les critiques constructives de l'auditoire, et en particulier celles de l'équipe de J.-P. Finance, ont fortement influencé la rédaction.

Publié avec le concours
de la Fondation Universitaire de Belgique
et de la Fondation Francqui

Table des matières

1 Introduction historique et philosophique à la théorie des ensembles **15**

1.1. La genèse de la notion de cardinal d'un ensemble 15
1.2. La définition cantorienne du cardinal d'un ensemble 18
1.3. La théorie naïve des ensembles 19
1.4. Les antinomies de la théorie naïve des ensembles 20
1.5. La théorie axiomatique des ensembles 21
1.6. Les axiomes de la théorie des ensembles 23
1.6.1. L'axiome de compréhension 23
1.6.2. L'axiome de séparation 24
1.6.3. L'axiome d'extensionnalité 25
1.6.4. L'axiome de la paire 26
1.6.5. L'axiome de la réunion 26
1.6.6. L'axiome de l'infini 26
1.6.7. L'axiome de la puissance 27
1.6.8. L'axiome de remplacement 28
1.6.9. L'axiome du choix 29
1.6.10. L'axiome de fondation 31
1.7. Ensembles non bien fondés 32
1.8. Antinomie et imprédicativité 35
1.9. La théorie des types de Russell 37
1.10. Théorie des types et théorie des ensembles 42
1.11. La métathéorie de la théorie des ensembles 45

2 Théorie des ensembles **49**
2.1. Introduction 49
2.2. Notion naïve d'ensemble 52
2.2.1. Extension et compréhension 53
2.2.2. Algèbre ensembliste 54
2.2.3. Algèbre relationnelle 56
2.2.4. Produit d'ensembles 60
2.2.5. Relation d'équivalence et ensemble quotient 62
2.2.6. Eléments 63
2.2.7. Théorie des ensembles et sémantique 63
2.3. Approche constructive 65
2.3.1. L'approche constructive concrète en informatique . . . 65
2.3.2. Représentation ensembliste d'objets mathématiques . . 66
2.3.3. Nombres naturels 68
2.3.4. Définitions récursives 72
2.3.5. Le théorème de von Neumann 80
2.3.6. Entiers relatifs 83
2.3.7. Nombres rationnels, réels et complexes 85
2.4. Approche formelle 88
2.4.1. Paradoxes 88
2.4.2. L'approche constructive abstraite en informatique . . . 90
2.4.3. Vers une théorie formelle 91
2.4.4. Construction axiomatique des ensembles 93
2.4.5. Construction des ensembles finis 94
2.4.6. Quelques théorèmes importants 96
2.4.7. Construction des ensembles infinis 98
2.4.8. Adéquation, complétude et indépendance 98
2.5. Ordinaux et axiome de fondation 99
2.5.1. Introduction 99
2.5.2. Naturels 100
2.5.3. Définition des ordinaux 100
2.5.4. Propriétés des ordinaux 104
2.5.5. Principe d'induction ordinale et récursivité ordinale . . 108
2.5.6. Applications du théorème de von Neumann 110
2.5.7. Modèles ensemblistes et axiome de fondation 113
2.5.8. Application de l'axiome de fondation 116
2.5.9. Ensembles non fondés 118
2.6. Cardinaux et axiome du choix 122
2.6.1. Equipotence 122
2.6.2. Théorème de Schröder et Bernstein 123
2.6.3. Exemples 125
2.6.4. Axiome du choix 126
2.6.5. Diverses formulations de l'axiome du choix 126

2.6.6. Application à la théorie des cardinaux 129
2.6.7. Application aux définitions récursives 131
2.6.8. Statut de l'axiome du choix 132
2.7. Théorie des ensembles et théorie des types 133
2.7.1. Approches alternatives au problème des types 133
2.7.2. Introduction à la théorie formelle des types 136

3 Logique pour programmes séquentiels 139
3.1. Introduction 139
3.2. Affectation 141
3.2.1. Introduction 141
3.2.2. Syntaxe d'une affectation 142
3.2.3. Sémantique opérationnelle d'une affectation 144
3.2.4. Sémantique relationnelle d'une affectation 145
3.2.5. Triplet de Hoare 145
3.2.6. Transformations associées à une affectation 147
3.2.7. Réduction des assertions à la logique classique 151
3.2.8. Affectation conditionnelle 153
3.3. Variables structurées 156
3.3.1. Tableaux 156
3.3.2. Enregistrements 158
3.3.3. Affectations multiples et structures composites 158
3.3.4. Application : réalisation d'une pile 159
3.4. Systèmes formels séquentiels 160
3.4.1. Définition 161
3.4.2. Sémantique relationnelle 163
3.4.3. Sémantique axiomatique 164
3.4.4. Transformations associées 167
3.4.5. Spécification et vérification des systèmes séquentiels . 169
3.5. Programmation séquentielle structurée 173
3.5.1. Vocabulaire 173
3.5.2. Syntaxe des programmes 174
3.5.3. Sémantique relationnelle des programmes structurés . 175
3.5.4. Logique de Hoare en programmation structurée 177
3.5.5. Terminaison 178
3.5.6. Un exemple élémentaire 179
3.6. Deux exemples classiques 182
3.6.1. Tri par insertion 182
3.6.2. Le programme "Find" de Hoare 184
3.6.3. Vérification assistée par ordinateur 192
3.6.4. Conclusion 196
3.7. Forme canonique des programmes impératifs 196
3.7.1. Introduction 196

3.7.2. Le non-déterminisme 197
3.7.3. Techniques de conversion 200
3.7.4. Affectations, conditions et procédures 201

4 Logique pour programmes parallèles 205
4.1. Introduction 205
4.2. Parallélisme et structuration 206
4.3. Parallélisme implicite : le tri systolique 208
4.3.1. Le problème 208
4.3.2. Le programme de base et son invariant 210
4.3.3. Comment trier une suite de nombres ? 212
4.3.4. Réalisation séquentielle 213
4.3.5. Communication synchrone 215
4.3.6. Une réalisation parallèle 215
4.3.7. Un point de technologie 216
4.3.8. Limitations pratiques du parallélisme implicite 217
4.3.9. Puissance théorique du parallélisme implicite 217
4.4. Systèmes formels concurrents 218
4.4.1. Introduction 218
4.4.2. Problème de l'exclusion mutuelle 219
4.4.3. Exclusion mutuelle entre deux processus 219
4.4.4. Un algorithme naïf 220
4.4.5. Une solution moins rigide 221
4.4.6. Problème du grain 222
4.4.7. L'algorithme de Peterson 224
4.5. Exclusion mutuelle entre n processus 225
4.5.1. Une solution élémentaire 227
4.5.2. Introduction d'un contrôleur 228
4.5.3. Une version plus réaliste 228
4.5.4. Propriétés du programme 230
4.5.5. Une version synchrone 231
4.5.6. Une version asynchrone 233
4.6. Exclusion mutuelle à contrôle réparti 234
4.6.1. Description informelle 235
4.6.2. Une première version abstraite 235
4.6.3. Un grain plus fin 238
4.6.4. Réalisation du schéma de priorité 241
4.6.5. Décomposition fine des transitions 241
4.7. Vers un système réel 242
4.7.1. Variables privées et transitions partagées 244
4.7.2. Une horloge distribuée 245
4.7.3. Evaluation locale des gardes 247
4.7.4. Réalisation en pseudo-CSP 248

4.7.5. Réalisation en CSP 249
4.7.6. Vérifications complémentaires 249
4.8. Calcul distribué d'un point fixe 256
4.8.1. Description du problème 256
4.8.2. Une solution abstraite 258
4.8.3. Une solution plus réaliste 258
4.8.4. Autres raffinements 262

5 Théorie du point fixe **265**
5.1. Introduction 265
5.1.1. Théories semi-formelles 265
5.1.2. Logique de déduction et logique d'expression 266
5.1.3. Logique typée 268
5.2. Théorie du point fixe 269
5.2.1. Introduction 269
5.2.2. Eléments remarquables d'un ensemble ordonné 270
5.2.3. La relation d'inclusion 274
5.2.4. Points fixes des fonctions croissantes 275
5.2.5. Points fixes des fonctions continues 278
5.2.6. Continuité en algèbre, en analyse et en topologie . . . 282
5.2.7. Points fixes dans un treillis 286
5.2.8. Exemples classiques 288
5.2.9. Propriétés des points fixes 288
5.3. Application à la programmation fonctionnelle 289
5.3.1. Introduction 290
5.3.2. Ensembles étendus 291
5.3.3. Programmation fonctionnelle du premier ordre 295
5.3.4. Evaluation de programmes fonctionnels 298
5.3.5. Conception de programmes fonctionnels 302
5.3.6. Démonstration automatique 305
5.3.7. Liens avec la programmation séquentielle 306
5.4. Application à la programmation impérative 309
5.4.1. La sémantique d'une boucle 309
5.4.2. Les invariants d'une boucle 310
5.5. Domaines sémantiques 316
5.5.1. Vers une sémantique des programmes fonctionnels . . . 316
5.5.2. Vers une sémantique des programmes impératifs 317
5.5.3. Exemples de domaines définis récursivement 318
5.5.4. Construction de domaines définis récursivement 319
5.6. Application à la programmation logique 322
5.6.1. Introduction 322
5.6.2. Nature des programmes logiques 323
5.6.3. Programmes logiques abstraits 324

5.6.4. Sémantique opérationnelle 324
5.6.5. Sémantique de point fixe 327
5.6.6. Mise en œuvre de la programmation logique 330

Bibliographie **337**

Index **345**

Avant-propos

L'ouvrage se veut introductif et ne suppose du lecteur que la connaissance préalable de la logique classique élémentaire, acquise par la lecture du premier volume, ou d'un autre traité de logique. Les références occasionnelles faites à un passage du premier volume sont précédées du chiffre romain I.

Seul le premier chapitre a vocation historique et a fait l'objet d'une étude bibliographique. Quand deux dates figurent à côté d'un nom, la première est celle de la parution, la seconde, celle de la version (réédition ou traduction) utilisée.

Chapitre 1
Introduction historique et philosophique à la théorie des ensembles

1.1. La genèse de la notion de cardinal d'un ensemble

D'Aristote à Leibniz, l'infini invoqué en mathématiques a été conçu comme potentiel (l'infini en puissance). Aristote dénie toute existence physique à l'infini, mais reconnaît que le mathématicien est conduit à faire appel à des grandeurs plus grandes ou plus petites que toute grandeur donnée. L'infini qui entre en jeu ici est un infini potentiel qui n'exige pas que l'on reconnaisse l'existence de totalités infinies achevées. Le concept d'infini potentiel s'accommode d'une définition négative. L'infini potentiel ne nous oblige pas vraiment à sortir du fini. Comme l'écrit Bolzano, "Une grandeur susceptible d'être toujours plus grande qu'on ne l'a d'abord supposée et de devenir plus grande que toute grandeur [finie] donnée, peut malgré tout rester constamment finie (Bolzano 1851, trad. 1993, p. 66)".

Le besoin de constituer positivement un concept mathématique d'infini apparaît, écrit H. Sinaceur, avec l'analyse galiléenne du mouvement, et surtout avec l'invention du calcul infinitésimal par Leibniz et Newton. Dans un premier temps, on se reconnaît le droit d'utiliser des quantités non finies dans des résultats exprimés en quantités finies, mais on croit pouvoir le faire sans devoir assumer ontologiquement l'existence de quantités infiniment petites. Leibniz confère à ces dernières le statut de *fictions*.

Cette vue prévaut encore chez Gauss qui déclare que "l'infini n'est qu'une façon de parler puisqu'on parle au sens propre de limites quand certains rapports diffèrent d'une quantité déterminée d'aussi peu qu'on veut tandis que d'autres rapports peuvent croître indéfiniment (Gauss 1831, cité par J. Breuer)". Le recours à la notion de limite est un des moyens d'esquiver

l'utilisation du concept d'*infini en acte*.

Dans *Philosophie mathématique*, Cavaillès reconnaît à Bolzano le mérite d'avoir réussi à "distinguer de l'infini improprement dit — l'infini de l'analyse qui ne concerne qu'une possibilité de variation et représente des nombres finis pris aussi grands qu'on veut — le véritable infini sur lequel jusque-là rien que de vague ou d'absurde n'avait été dit (Cavaillès 1962, p. 67)".

On peut illustrer par un exemple très simple, qui ne fait pas appel au calcul infinitésimal, la manière dont le concept de limite peut remplacer le concept d'infini. Considérons le segment de droite OX constitué par l'hypoténuse d'un triangle rectangle dont les côtés mesurent l'unité. En vertu du théorème de Pythagore, ce segment a une longueur de $\sqrt{2}$. Cette longueur n'est pas rationnelle, mais est la limite d'une suite (infinie) d'approximations rationnelles.

Si le concept d'infini en acte a mis si longtemps à émerger, c'est que, pour constituer ce concept, il a fallu renverser un obstacle épistémologique majeur : la croyance obstinée à l'axiome selon lequel le tout est plus grand que la partie. Il vaut la peine de rappeler le cheminement laborieux qui a conduit Bolzano, un mathématicien philosophe de langue allemande, à renverser cet obstacle dans un petit livre publié à titre posthume : *Paradoxien des Unendlichen* (1851).

Un des maîtres de l'Ecole néoplatonicienne, Proclus (412-485), avait remarqué dans son commentaire du premier livre des *Eléments* d'Euclide que l'ensemble des diamètres d'un cercle est "deux fois plus petit" que l'ensemble des demi-cercles. En 1638, Galilée compare l'ensemble des nombres entiers à l'ensemble de leurs carrés et envisage, à titre d'hypothèse, la possibilité de dire de ces ensembles qu'ils sont égaux, mais il ne la retient pas et renonce à l'application des relations d'égalité ou d'ordre aux ensembles infinis.

Leibniz, à son tour, se penche sur la possibilité d'établir une bijection entre un ensemble infini et une de ses parties. Dans une lettre à Malebranche, il signale qu'à n'importe quel nombre donné correspond le nombre qui est son double. Mais il fait un pas de plus et entrevoit l'idée d'associer un nombre à ces ensembles infinis que sont l'ensemble des naturels et l'ensemble des naturels pairs, comme on peut associer un nombre à un ensemble fini (Leibniz entrevoit donc l'idée de cardinal d'un ensemble qui sera introduite vers 1870 par Cantor).

Mais Leibniz voit très clairement que si l'on pose l'existence d'"un nombre de tous les nombres" qui serait à la fois le nombre de l'ensemble infini des naturels et celui de l'ensemble infini des naturels pairs, on doit renoncer à l'axiome "le tout est plus grand que la partie". Dans une lettre à Jean Gallois, il écrit : "Puisque donc dans ce nombre infini il y aurait autant de nombres pairs que de nombres pairs et impairs pris ensemble, il suit que pour ce nombre infini, ce grand axiome serait réfuté que le tout est plus grand que la partie... mais il est impossible que ce grand axiome soit réfuté... le nombre infini est donc impossible, il n'est pas un tout, il n'est rien (Leibniz 1672, 1923-, p. 226)".[1]

[1] Nous remercions Mme L. Bouquiaux, D.Ph., de nous avoir signalé ce passage.

Leibniz voit une autre objection à l'admission de l'existence d'un nombre infini : celle-ci nous conduirait à une conséquence absurde, la reconnaissance qu'il existe un nombre qui est le plus grand nombre. Comme l'écrit H. Sinaceur, "Leibniz, qui a tant fait pour le progrès du calcul infinitésimal et tant manié les différents ordres d'infiniment petits, supposait devoir en rester à l'unicité dans le domaine de l'infiniment grand; d'où l'absurdité : s'il y avait un nombre infiniment grand, il serait *le plus grand nombre*; conclusion : il n'y a pas de nombre infini (Introd. à Bolzano 1993, pp. 22-23)".

Dans *Les Paradoxes de l'Infini*, Bolzano va lever les deux obstacles qui empêchaient Leibniz de reconnaître l'infini en acte et de lui associer un nombre infini; en premier lieu, il va dissiper l'apparence de paradoxe que présentent les ensembles qui peuvent être appariés à une de leurs parties et y voir, au contraire, une propriété qui caractérise positivement les ensembles infinis : "Nous devons considérer, à présent, écrit-il, une propriété très remarquable qui peut se rencontrer, qui, à vrai dire, se rencontre toujours dans le rapport de deux ensembles, *quand ils sont tous deux infinis* : d'une part, il est possible d'apparier terme à terme tous les éléments de ces deux ensembles... D'autre part, il est pourtant possible que l'un des deux ensembles comprenne l'autre comme *simple partie*... (Bolzano, *op. cit.*, p. 85)". En second lieu, Bolzano va reconnaître l'existence d'une multiplicité d'infinis : "Quand on accorde qu'il y a des pluralités infinies, et donc aussi, en général, des grandeurs infinies, on ne peut mettre en doute l'existence de grandeurs infinies distinctes... (Bolzano, p. 102)".

Reconnaître que deux ensembles dont l'un est un sous-ensemble propre de l'autre peuvent être reliés par une bijection est une chose, dire qu'ils contiennent l'ensemble des mêmes termes en est une autre qui exige de renoncer à l'axiome qui affirme que le tout est plus grand que la partie. Dans un passage de son livre — mais, comme le note H. Sinaceur, ce passage est unique — Bolzano assume cette conséquence. Comparant la somme infinie des carrés des entiers naturels (S1) à la somme infinie de ces mêmes entiers (S2), il écrit : "Nul n'ira contester... que beaucoup de termes, et même une infinité, apparaissent dans S2 sans figurer dans S1... ". Et il poursuit : "S1, la somme des carrés, n'est pourtant pas plus petite mais incontestablement plus grande que S2 ... Malgré toute apparence du contraire, les deux séries (non encore considérées comme sommes...) comportent en effet le même *ensemble de termes* (Bolzano, p. 13)".

Pour aller au bout de sa découverte, Bolzano aurait dû élargir la notion d'égalité et la dissocier de celle d'identité de manière à pouvoir reconnaître que deux ensembles infinis, comme par exemple l'ensemble des naturels et l'ensemble de leurs carrés, peuvent être "égaux" (en taille) sans être identiques. Bolzano n'ira pas jusque là. Il a frôlé l'élargissement du concept d'égalité, mais c'est à Cantor que revient le mérite d'avoir forgé le concept d'équipotence qui consacre cet élargissement.

H. Sinaceur a bien mis ce point en lumière. Faisant la part de ce qui est novateur et de ce qui est repris à la tradition chez Bolzano, elle écrit à propos de ce dernier: "D'une part, il montre que, loin d'être un paradoxe, l'existence d'une bijection d'un ensemble sur une de ses parties propres est une *caractéristique* des ensembles infinis. Mais d'autre part, il n'arrive pas à concevoir une relation d'égalité et une relation d'ordre qui soient, dans l'infini, définies *autrement* que dans le fini". Elle poursuit en précisant en ces termes l'apport de Cantor: "... Cantor a défini l'équipotence (*Gleichmächtigkeit*) de deux ensembles infinis par l'existence d'une bijection de l'un sur l'autre. Deux ensembles équipotents ont le même *nombre* cardinal. Et cette définition vaut aussi bien pour le fini que pour l'infini. La "puissance" d'un ensemble infini est donc l'extension au cas des ensembles infinis du concept de nombre fini, et l'équipotence l'extension de la relation d'égalité (Bolzano 1993, notes, pp. 92-93)".

Rappelons que c'est Dedekind qui transformera le paradoxe des ensembles infinis (la propriété surprenante qu'ils ont de pouvoir être mis en bijection avec une de leurs parties propres) en définition de ces ensembles.

1.2. La définition cantorienne du cardinal d'un ensemble

C'est dans une lettre de 1884 à Lasswitz citée dans Cavaillès (*op. cit.*, p. 97) que l'on trouve la définition précitée: "J'entends par puissance ou nombre cardinal d'un ensemble M, le concept universel ou générique que l'on obtient en faisant abstraction pour l'ensemble aussi bien de la constitution de ses éléments que de toutes les relations que ces éléments ont entre eux et ne considère que ce qui est commun à tous les ensembles *équivalents* à M".

D. Vernant voit, à juste titre, dans cette première tentative de définition du nombre, un acquis définitif. En introduisant la notion d'équivalence fondée sur l'appariement des éléments de plusieurs ensembles, cette définition, écrit-il, "rompt radicalement avec la tradition qui prétendait dériver le concept de nombre de l'opération 'empirique' de dénombrement (Vernant 1993, p. 125)". Cette rupture est un progrès. L'opération concrète de dénombrement ne peut pas servir à définir le nombre car elle présuppose le nombre, et, qui plus est, le nombre ordinal.

Dans *The Principles of Mathematics* (1903), Russell a mis en évidence le caractère circulaire de la définition traditionnelle de nombre reposant sur l'opération concrète de dénombrement: "Quand nous disons un, deux, trois, etc., nous devons nécessairement considérer quelque relation une à une entre les nombres utilisés pour compter et les objets comptés ... ainsi nous relions une classe d'objets à une classe de nombres; et la classe de nombres consiste en tous les nombres de 1 à quelque nombre n (Russell 1903, p. 133)".

Un autre avantage de la définition cantorienne de nombre cardinal réside dans le fait qu'"elle autorise l'extension du concept de nombre aux *ensembles*

infinis puisque, s'il est exclu d'énumérer de tels ensembles, il demeure possible de s'assurer de leur équivalence en appariant leurs éléments (Vernant, *op. cit.*, p. 125)". En d'autres termes, la définition cantorienne ouvre la voie à un véritable calcul de l'infini.

1.3. La théorie naïve des ensembles

Cantor a créé une théorie qu'on pourrait décrire comme une arithmétique des ensembles infinis. Il reconnaît des degrés dans l'infini. Aux ensembles infinis, il associe des nombres nouveaux, les nombres transfinis qui donnent une idée de la taille de l'ensemble. Plus l'ensemble contient d'éléments, plus le nombre associé est grand. A l'ensemble des entiers naturels est associé un nombre transfini qu'il notera $\aleph_0$ (lire "aleph 0"). C'est le premier des "nombres transfinis" qui forment, eux-mêmes, une suite infinie. Ces nombres transfinis étant associés à des ensembles, il devient possible grâce à eux de calculer avec les ensembles.

Le nombre associé à un ensemble est le "cardinal" de cet ensemble ou sa "puissance" et, tout naturellement, le cardinal d'un ensemble fini se confond avec le nombre d'éléments de cet ensemble. Le nombre cardinal d'un ensemble, c'est "le concept général auquel on arrive... quand on fait abstraction de la nature des divers éléments m d'un ensemble M et de l'ordre dans lequel ils sont donnés (Cantor 1895, 1955, p. 86)". Les notions nouvelles de nombre cardinal ou de puissance d'un ensemble s'appliquent aussi bien aux ensembles finis qu'aux ensembles infinis. La théorie des ensembles donne ici toute la mesure de sa puissance unificatrice: il n'existe plus, comme l'écrit Breuer, de barrière fondamentale entre le fini et l'infini.

Cantor définit pour les nombres transfinis des opérations qui jusque là ne s'appliquaient qu'aux nombres finis: l'addition, la multiplication, l'exponentiation. Les axiomes nouveaux que formule Cantor se recommandent en raison de leur évidence intuitive et de leur fécondité. Utilisés conjointement avec la notion de correspondance biunivoque empruntée par Cantor à Dedekind, ces axiomes permettent de démontrer des théorèmes surprenants, comme l'équipotence (l'identité de puissance) entre l'ensemble des nombres réels de l'intervalle $[0, 1]$, l'ensemble des points d'un plan, l'ensemble des points d'un cube et l'ensemble des points d'un espace tridimensionnel illimité (Breuer 1961).

Cantor introduit la notion de "type ordinal d'un ensemble [bien] ordonné" et définit à partir de celle-ci la notion de nombre ordinal (la caractérisation cantorienne du bon ordre n'est pas tout à fait celle classiquement admise aujourd'hui mais elle lui est équivalente, comme le montre M. Hallett (1984, pp. 51-52)). A chaque ensemble [bien] ordonné est associé un type d'ordre: "... ce type est le concept général qui résulte de (l'ensemble) M si nous faisons abstraction de la nature de ses éléments en ne retenant que leur ordre en sorte que de ces derniers procèdent des unités qui se trouvent les unes par rapport

aux autres dans une relation de succession (Cantor 1897, 1955, pp. 151-152)". A l'aide de la notion de type ordinal, on définit ensuite la notion de nombre ordinal: "tous les bons ordres qui sont semblables l'un à l'autre (c'est-à-dire dont les membres respectifs sont reliés par une relation biunivoque respectant l'ordre) et eux seuls ont le même type ordinal. Nous appelons le type ordinal d'un ensemble bien ordonné F son 'nombre ordinal' (Cantor, *ibid.*)".

1.4. Les antinomies de la théorie naïve des ensembles

Nous mentionnerons en premier l'antinomie de Burali-Forti touchant la théorie des ordinaux. Exposée en 1897 par l'auteur précité, elle était connue de Cantor qui selon le témoignage de Bernstein en a eu l'idée dès 1895. Cavaillès la résume en ces termes: "l'ensemble de tous les nombres ordinaux ... rangés par ordre de grandeur forme un ensemble bien ordonné, qui a donc pour type un ordinal a; mais celui-ci ne peut figurer dans la suite W [des ordinaux] car à son rang il y dénoterait un segment de type a (l'ensemble de tous les ordinaux le précédant) et W serait semblable à un de ses segments (Cavaillès 1962, p. 116)".

La théorie des ordinaux n'est pas la seule à être menacée par une antinomie. La théorie des cardinaux l'est aussi. Cantor a démontré que le cardinal de l'ensemble des parties d'un ensemble est plus grand que le cardinal de l'ensemble lui-même. Considérons le cardinal de l'ensemble de tous les ensembles. En vertu du théorème précité, l'ensemble des parties de cet ensemble doit avoir un cardinal plus grand que le précédent. Mais alors l'ensemble de tous les ensembles n'est pas l'ensemble de tous les ensembles. Nous arrivons à une contradiction.

Face à cette situation, Cantor affine sa conception de ce qu'est un ensemble. Alors qu'en 1895, il appelait ensemble un "rassemblement en un tout d'objets déterminés et bien distincts de notre intuition ou de notre pensée", en 1899, dans une lettre à Dedekind, il introduit une distinction entre multiplicités inconsistantes, qui ne sont pas des ensembles, et multiplicités consistantes, qui en sont: "En partant du concept d'une multiplicité déterminée d'objets (un système, une classe), il m'est apparu nécessaire de distinguer deux sortes de multiplicités ... En effet, une multiplicité peut être ainsi constituée que l'hypothèse d'une 'existence simultanée' de *tous* ses éléments conduit à une contradiction, de telle sorte qu'il est impossible de concevoir cette multiplicité comme une unité, comme 'un objet achevé'. Je nomme de telles multiplicités 'mutiplicités absolument infinies' ou 'inconsistantes' ... Si, au contraire, la totalité des éléments d'une multiplicité peut être pensée comme existant simultanément, de telle sorte qu'il soit possible de la concevoir comme *un seul objet*, je la nomme une 'multiplicité consistante' ou un 'ensemble' (Cavaillès 1962, p. 239)".

Cantor revient sur la question un mois plus tard dans une lettre à Dedekind: "On doit se demander, écrit-il, d'où je sais que les multiplicités bien ordonnées

ou suites auxquelles j'assigne les nombres cardinaux $\aleph_0$, $\aleph_1$, ..., $\aleph_{\omega_0}$, $\aleph_{\omega_1}$, sont réellement des 'ensembles' dans le sens du mot qui a été expliqué, c'est-à-dire des 'multiplicités consistantes'. La question, écrit Cantor, peut être étendue également aux multiplicités finies. Dans les deux cas, sa réponse est la même: *il n'y a pas* de démonstration de leur 'consistance'. En d'autres termes, le fait de la 'consistance' des multiplicités finies est une vérité simple, indémontrable, c'est 'l'*axiome* de l'arithmétique' (dans l'ancien sens du mot). Et, de même, la 'consistance' des multiplicités auxquelles j'attribue comme nombres cardinaux les $\aleph$, c'est 'l'axiome de l'arithmétique transfinie élargie' (Cavaillès, *ibid.*, p. 244)".

1.5. La théorie axiomatique des ensembles

Cantor a constitué une théorie nouvelle pour répondre à des problèmes qui se posaient dans la mathématique existante. Il a, comme l'écrit Cavaillès, forgé des notions et des procédés qui permettent d'obtenir des résultats indépendants de la structure particulière des objets mathématiques dont il est parti. Mais le concept-clef de cette théorie abstraite, la notion de base, c'est-à-dire la notion d'ensemble, donne naissance à des contradictions. Pour éviter les contradictions, Cantor utilise la technique bien connue des mathématiciens: il "remodèle", comme l'écrit D. Vernant, son axiomatique, mais il s'arrête à une axiomatique au sens ancien: les principes sont reçus comme tels en raison de leur évidence intrinsèque.

C'est à Zermelo que revient le mérite d'avoir appliqué à la théorie des ensembles l'axiomatisation au sens moderne du terme, suivant en cela l'exemple donné par Hilbert dans les *Grundlagen der Geometrie.* Reconnaissant qu'on n'avait pas encore, au moment où il écrivait, trouvé d'alternative à la définition explicite du terme "ensemble" donnée par Cantor, Zermelo concluait: "Dans ces conditions, il n'y a actuellement pas d'autre issue que de s'engager dans la direction opposée: chercher, à partir de la théorie des ensembles telle qu'elle est donnée historiquement, quels sont les principes requis pour fonder cette discipline mathématique. Il faut pour cela restreindre suffisamment les principes de façon à exclure toute contradiction, tout en les gardant cependant suffisamment larges, afin que tout ce qu'il y a de précieux dans cette théorie soit conservé (Zermelo 1908, 1992, pp. 370-371)".

Chez Zermelo, les axiomes ne sont pas choisis pour leur évidence intrinsèque, mais parce qu'on peut en déduire les acquis de la théorie et qu'on peut y voir des définitions implicites de la notion de base: "... entre les objets indéterminés qui constituent le champ de la théorie ... l'axiomatique définit par ses propriétés une relation fondamentale $\in$ élément-ensemble. Les seconds termes seront les ensembles, ou plutôt la théorie ne s'occupe que de déduire logiquement de nouvelles propriétés à partir des propriétés initiales de la relation, sans s'inquiéter de la réalité ou de la structure des termes (Cavaillès 1962, p. 143)".

Il importe ici d'être attentif à une distinction essentielle, soulignée par Cocchiarella, entre la notion mathématique d'ensemble définie à partir de la relation d'appartenance et la notion logique de classe conçue comme l'extension d'un concept. Quand on a en vue la notion logique de classe, la notion de concept est primitive et celle d'appartenance, dérivée: "Soit P un concept et C son extension. On dira qu'un objet x est membre de la classe C si et seulement si x tombe sous le concept C (Shapiro 1991, p. 20)". Quand on a en vue la notion mathématique d'ensemble, c'est la notion d'appartenance qui est première.

Longy souligne bien le fait que la théorie des ensembles est purement extensionnelle: "... Zermelo pose axiomatiquement l'existence de certains ensembles particuliers (axiomes II et VII) et de règles permettant de créer de nouveaux ensembles à partir d'ensembles déjà existants (axiomes III à VI). Ce qui est abandonné dans une telle détermination du concept d'ensemble, c'est principalement l'idée qu'à chaque propriété peut être associé l'ensemble de tous les objets la possédant (Introd. à Zermelo 1908, 1992, p. 368)".

Une fois que l'on dispose de l'axiomatique de Zermelo, on peut dire qu'un ensemble, c'est n'importe quoi qui rend vrais les axiomes en question. Mais il ne faudrait pas en conclure que ces axiomes ont été choisis au hasard. Ils visent, au contraire, à représenter une certaine conception pré-axiomatique de l'ensemble. A cette conception qui diffère, comme nous l'avons vu, de la classe familière aux logiciens, on a donné le nom de conception itérative de l'ensemble.

Le concept itératif d'ensemble est celui avec lequel travaille, en fait, la théorie naïve des ensembles de Cantor. Il est parfaitement consistant et ne donne pas naissance à des contradictions. (Ce qui donne naissance à des contradictions, ce n'est pas le concept d'ensemble mais c'est la définition explicite de l'ensemble qu'on trouve dans la théorie naïve.) Telle est notamment la position de Gödel: "il suit immédiatement (de l'explication du concept d'ensemble en termes d'itération de l'opération 'ensemble de') qu'un ensemble de tous les ensembles ou autres ensembles d'une extension similaire ne peuvent exister, puisque chaque ensemble obtenu de cette manière donne immédiatement lieu à une application ultérieure de l'opération 'ensemble de' et donc à l'existence d'ensembles plus grands (Gödel 1947, 1990, p. 519, note)".

Nous allons à présent passer en revue et commenter les axiomes de la théorie des ensembles. L'axiomatique présentée ici est celle de Zermelo-Fraenkel telle qu'on la trouve exposée dans *Foundations of Set Theory* de Bar-Hillel, Fraenkel et Levy (1973). Dans l'article fondateur de 1908, Zermelo a montré comment la théorie entière de Cantor et de Dedekind pouvait "être ramenée à quelques définitions et à sept 'principes' ou 'axiomes' (Zermelo 1908, 1992, p. 371)". Fraenkel, de son côté, a introduit trois innovations.

La première de ces innovations réside dans la transformation du matériel de la théorie: "la notion d'un champ général des objets de la théorie, parmi lesquels les ensembles devraient être distingués, est abandonnée; on exclut tous

les objets non-ensembles (à l'exception de l'ensemble vide). Seuls les ensembles sont considérés, et, d'abord, de façon hypothétique. Il n'y a pour cette théorie aucun axiome d'existence inconditionnée. L'axiome de *détermination* devient : 'si a, b et A sont des ensembles et si $a = b$ et a est membre de A, on a aussi b est membre de A' (Cavaillès 1962, p. 146)".

La deuxième différence majeure réside dans la formulation de l'axiome de séparation dont nous parlerons plus loin. Sous la forme qu'il avait chez Zermelo, cet axiome restreignait la possibilité de définir un ensemble à partir d'une propriété par les deux exigences suivantes : d'une part, on doit partir d'un ensemble déjà donné; d'autre part, la propriété doit être "bien définie". Cette seconde exigence est vague. Elle sera précisée par Skolem : "... une propriété sera dite bien définie, si elle s'exprime dans le langage du calcul des prédicats du premier ordre et ne fait intervenir, outre les constantes logiques, que les deux symboles de prédicats $\in$ et $=$ (Longy, Introd. à Zermelo 1908, 1992, p. 369)".

La troisième innovation de Fraenkel est l'introduction d'un axiome nouveau, l'axiome de remplacement, dont il sera question plus loin.

1.6. Les axiomes de la théorie des ensembles

1.6.1. *L'axiome de compréhension*

L'idée familière qu'à chaque propriété ou à chaque concept correspond une classe, la classe des objets qui possèdent la propriété ou tombent sous le concept, doit être abandonnée parce qu'elle conduit tout droit à une antinomie découverte simultanément par Zermelo et par Russell mais publiée en premier lieu par Russell en 1901. On arrive à cette antinomie en utilisant l'argument connu sous le nom de "diagonalisation" qui intervient dans la démonstration du théorème de Cantor affirmant que l'ensemble des parties d'un ensemble (fini ou infini) est strictement plus grand que l'ensemble lui-même. La contradiction survient dès que l'on affirme qu'il existe un ensemble des ensembles qui correspond à la propriété exprimée par la fonction propositionnelle "x n'est pas membre de x".

Appelons axiome de compréhension le principe affirmant qu'à toute fonction propositionnelle Fx (phrase ouverte) correspond un ensemble, à savoir l'ensemble des êtres qui satisfont la fonction propositionnelle en question. Formellement, ce schéma d'axiome s'énonce comme suit :

$$\exists y \forall x (x \in y \equiv Fx).$$

Substituons au schéma de prédicat Fx le prédicat $x \notin x$. Instancions en c les variables y, puis x. Nous arrivons à la formule contradictoire

$$c \in c \equiv c \notin c.$$

On rapproche quelquefois le paradoxe des ensembles qui ne se contiennent pas eux-mêmes de l'énoncé qui affirme l'existence d'un barbier qui rase tous et

seulement ceux qui ne se rasent pas eux-mêmes. Soit b ce barbier; l'énoncé qui lui attribue la propriété précitée s'écrit comme suit :

$$\forall x\,(bRx \equiv \neg xRx)\,.$$

Ici aussi, on obtient une contradiction en instanciant x en b. De ce que l'hypothèse de l'existence d'un tel barbier conduit à une contradiction, on conclut simplement qu'un tel barbier n'existe pas. Nous nous trouvons ici devant une preuve par l'absurde. Dans le cas du paradoxe découvert par Zermelo et Russell, la situation est différente. Ce qui est mis en cause, c'est un principe fondamental auquel on ne peut renoncer. Deux solutions seront proposées : celle de Zermelo qui, par l'axiome de séparation, limite le principe en cause et celle de Russell qui empêche la contradiction de se produire en modifiant non la théorie, mais le langage de celle-ci au moyen de la théorie des types.

1.6.2. *L'axiome de séparation*

En 1908, Zermelo formule l'axiome de séparation (*Axiom der Aussonderung*) en ces termes : "Si l'énoncé (la condition Fx) est défini pour tous les membres d'un ensemble M, alors M a toujours un sous-ensemble qui contient tous et seulement les membres de M pour lesquels Fx est vrai", ce qui formellement s'écrit :

$$\forall a \exists y \forall x\,(x \in y \equiv (x \in a \wedge Fx))\,.$$

L'axiome de séparation, qui est une version affaiblie de l'axiome de compréhension, ne permet plus d'affirmer l'existence d'un ensemble dès que l'on possède une "condition", c'est-à-dire une phrase ouverte. Il affirme seulement que si l'on a un ensemble, au départ, et une phrase ouverte Fx, on peut affirmer l'existence d'un sous-ensemble d'êtres qui vérifient la condition Fx.

L'axiome de séparation évite, comme on va le voir, le paradoxe russellien des ensembles qui ne se contiennent pas eux-mêmes. Donnons-nous l'ensemble de départ a et substituons à Fx la phrase ouverte $x \notin x$. L'axiome de séparation nous permet d'affirmer l'existence d'un ensemble b qui est l'ensemble des x qui sont membres de a et qui ne sont pas membres d'eux-mêmes; on obtient

$$\forall x\,(x \in b \equiv (x \in a \wedge x \notin x))\,.$$

Instantions x en b et nous obtenons :

$$b \in b \equiv (b \in a \wedge b \notin b)\,.$$

Cet énoncé n'est nullement contradictoire; il se réduit à :

$$b \notin b \wedge b \notin a\,.$$

On arrive à la conclusion non triviale que pour tout ensemble a, il y a quelque chose qui n'y est pas contenu comme membre, en d'autres termes, rien ne contient tout.

Il y a certes un ensemble universel. Il suffit, pour l'obtenir, de se donner au départ un ensemble a et une phrase ouverte appropriée, par exemple, la phrase ouverte $x = x$, puis d'utiliser l'axiome de séparation. L'ensemble que l'on obtient ainsi est bien l'ensemble universel, mais universel relativement à a, et non pas absolument universel. L'ensemble universel qu'on a obtenu de la sorte nous suffira pour définir ensuite la notion de complément d'un ensemble, mais comme l'ensemble universel U qu'on vient de produire est un sous-ensemble de a et que a n'est pas absolument universel, a fortiori U ne l'est-il pas. Comme il n'y a pas d'ensemble absolument universel qui contiendrait tous les ensembles, il n'y a pas non plus d'ensemble qui contiendrait tous les ensembles qui ne sont pas membres de a; en d'autres termes l'ensemble a de départ n'a pas de complément. En effet, si a avait un complément, la réunion de a avec ce complément (réunion dont l'existence est garantie par un autre axiome de la théorie ZF) constituerait l'ensemble absolument universel, dont on a prouvé l'inexistence.

1.6.3. *L'axiome d'extensionnalité*

L'axiome d'extensionnalité affirme que des ensembles x et y qui ont les mêmes membres sont identiques, ou encore $x = y$ si, pour tout z, z est membre de x si et seulement si z est membre de y. Formellement :

$$\forall x \forall y \, (\forall z \, (z \in x \equiv z \in y) \; \Rightarrow \; x = y) \, .$$

Le critère d'identification des ensembles ou des classes — à ce stade, nous ne distinguons pas les premiers des secondes — est beaucoup moins exigeant que le critère d'identification des propriétés. Par exemple, la classe des trilatères est identique à celle des triangles, bien que la propriété exprimée par la phrase ouverte "x est trilatéral" diffère de celle de "x est triangulaire". Cet axiome permet de prouver par exemple qu'il n'existe qu'un seul ensemble vide, que celui-ci soit créé à partir de l'ensemble a au moyen de la phrase ouverte "$x \neq x$" ou de la phrase ouverte "x est pair et impair", bien que les propriétés exprimées par ces deux phrases ouvertes soient différentes. La théorie des ensembles est purement extensionnelle, ce qui lui confère une transparence qu'elle n'aurait pas si elle opérait avec des notions intensionnelles comme celle de propriété. En effet, on manque d'un critère qui nous dise quand deux propriétés sont identiques.

Il y a cependant des circonstances où le rejet des notions intensionnelles nous ferait perdre quelque chose. Considérons, par exemple, la notion de fonction. "Selon le point de vue ensembliste ou extensionnel, écrit M. Crabbé, la fonction est identifiée à son graphe (Crabbé 1986, p. 63)". Par exemple la fonction carré définie sur les entiers naturels sera simplement l'ensemble des paires ordonnées $\{(0,0),(1,1),(2,4),(3,9),\ldots\}$. Or il existe des disciplines où un concept plus fin de la fonction est nécessaire. Le théoricien des langages de programmation, par exemple, préfère une notion plus fine de la fonction,

qui prend en compte le type de travail effectué par l'application de celle-ci à son argument. Les deux symboles fonctionnels $x\,(x+1)$ et x^2+x seront vus comme exprimant des fonctions différentes bien que les fonctions en question aient le même graphe et soient identiques au point de vue de la théorie des ensembles si on ne prend en compte que leur graphe. Mais dira-t-on qu'il y a autant de fonctions calculables que de programmes ? Ce n'est pas acceptable. Quel est alors le critère d'identification de deux fonctions au sens intensionnel du terme ? On voit bien ici le caractère fuyant de l'intension.

1.6.4. *L'axiome de la paire*

Si deux ensembles a et b existent, alors il existe un ensemble y qui contient exactement a et b, ce qui se note ainsi :

$$\forall a \forall b \exists y \forall x\,(x \in y \;\equiv\; (x = a \vee x = b))\,.$$

Cet axiome permet, à partir des ensembles a et b, d'engendrer l'ensemble $\{a, b\}$, puis, récursivement, les ensembles $\{\{a, b\}, a\}$, $\{\{a, b\}, b\}$, etc., à l'infini. Il permet aussi de former l'ensemble $\{a, a\}$ qui, en vertu du principe d'extensionnalité, se réduit au singleton $\{a\}$.

1.6.5. *L'axiome de la réunion*

Pour tout ensemble a, il existe un ensemble dont les membres sont exactement les membres des membres de a, ce qui se note :

$$\forall a \exists y \forall x\,(x \in y \;\equiv\; \exists z\,(x \in z \wedge z \in a))\,.$$

1.6.6. *L'axiome de l'infini*

L'axiome de l'infini affirme l'existence d'un ensemble contenant l'ensemble vide et tel que si x en est membre, $\{x\}$ l'est aussi, ce qui se note :

$$\exists z\,[\emptyset \in z \wedge \forall x\,(x \in z \Rightarrow \{x\} \in z)]\,.$$

Cet ensemble est infini : il contient les éléments $\emptyset, \{\emptyset\}, \{\{\emptyset\}\}$, etc., tous distincts. Le but de cet axiome est d'introduire au moins un ensemble infini, ce que les axiomes précédents ne permettaient pas de faire; la nature exacte de cet ensemble n'est pas essentielle.

On s'est parfois demandé si la théorie des ensembles était une partie de la logique ou une partie des mathématiques distincte de la logique. L'axiome de l'infini est, par excellence, une proposition qu'on peut invoquer à l'appui de la seconde solution. On peut certainement souscrire à l'affirmation suivante faite par Russell dans *Introduction to Mathematical Philosophy* (1919) : "L'infini n'est pas une notion logiquement contradictoire, mais son existence n'est pas non plus logiquement démontrable : nous devons en conclure que nous ne pouvons savoir *a priori* si le nombre de choses dans le monde est fini ou infini

(Russell 1919, 1991, p. 271)". Remarquons que Zermelo ne tranche nullement par un axiome mathématique une question de physicien ou de cosmologiste concernant la finitude du monde physique. En effet, le matériau dont est constitué l'ensemble infini dont il postule l'existence, c'est l'ensemble vide.

Affirmer l'existence d'un ensemble infini, c'est admettre l'infini en acte. Telle était bien l'intention de Cantor qui affirmait qu'à tout infini en puissance correspond un infini en acte. Prenons, par exemple, le domaine **N** des naturels. La position de Cantor n'est pas de dire que nous sommes capables de saisir cette totalité par intuition. Cantor ne nie pas que nous ne pouvons appréhender l'ensemble des nombres naturels qu'au travers de l'opération d'ajouter une unité indéfiniment, mais il affirme qu'en dépit de cette appréhension imparfaite de l'intension du prédicat "nombre naturel", "l'extension ... *existe déjà en totalité* ... Sans égard pour notre capacité limitée de construire ou d'engendrer les nombres naturels, *il existe* une somme complète de tels nombres. Nos limitations constructives ne sont pas pertinentes (Hallett 1984, p. 27)".

Deux positions philosophiques s'affrontent : d'une part, il y a les partisans de la thèse constructiviste : "pas de collection sans collectionneur !"; d'autre part, il y a la thèse réaliste à laquelle Gödel, notamment, souscrit pleinement : "Les classes et les concepts peuvent être conçus comme des objets réels, les classes seraient des 'pluralités de choses' ou des structures consistant en pluralités de choses, tandis que les concepts sont conçus comme existant indépendamment de nos définitions et constructions (Gödel 1944, 1990, p. 137)". Gödel étaie sa position par l'argument suivant : "Il me semble que supposer l'existence de tels objets est aussi légitime que de supposer l'existence de corps physiques et qu'il y a tout autant de raisons de croire à leur existence. Les classes et les concepts sont aussi nécessaires à l'obtention d'un système satisfaisant de mathématiques que les corps physiques sont nécessaires à l'obtention d'une théorie satisfaisante à propos de nos perceptions sensibles (Gödel, *ibid.*)".

1.6.7. *L'axiome de la puissance*

Pour tout ensemble a, il existe un ensemble qui a pour membres exactement les sous-ensembles de a. Ceci se note :

$$\forall a \exists y \forall x \, (x \in y \equiv x \subset a) \, .$$

Soit l'ensemble **N**. Cet ensemble a des sous-ensembles, parmi lesquels l'ensemble des x qui sont membres de **N** et impairs. Quelle raison avons-nous de poser l'existence de l'ensemble de tous les sous-ensembles de **N** ? Les recherches de Zermelo sur le problème des ensembles bien ordonnés montrent qu'il est indispensable de poser l'existence de l'ensemble des parties (sous-ensembles) de a dès que nous admettons l'existence de a et d'une relation définie sur a.

L'argument sur lequel repose cette affirmation est le suivant : "Dans (la) conception (de Zermelo), une relation d'ordre sur a est une collection de sous-ensembles de a ordonnés par la relation d'inclusion. Si cette collection doit être

un objet, elle doit être un ensemble et la seule manière de montrer que c'est un ensemble, c'est de la séparer (par l'axiome de séparation) de la collection de tous les sous-ensembles de a, ce qui signifie que cette dernière collection doit elle-même être un ensemble (Hallett 1984, p. 212)".

L'axiome de la puissance cautionne le recours aux définitions imprédicatives qui seront examinées plus loin, définitions dont on a besoin pour formuler et démontrer des propositions fondamentales de l'analyse telles que le principe affirmant que les ensembles majorés de nombres réels ont une plus petite borne supérieure.

Notons que l'axiome de la puissance ne découle pas de la notion intuitive d'ensemble itératif. En introduisant cet axiome, Zermelo enrichit délibérément la notion intuitive d'ensemble.

1.6.8. *L'axiome de remplacement*

Les axiomes de Zermelo mentionnés jusqu'ici, auxquels on ajoute l'axiome du choix dont il sera question plus loin, suffisent à la démonstration des propositions de base de l'arithmétique et de l'analyse. En revanche, ils ne suffisent plus dès que l'on désire prouver l'existence d'une limite $\aleph_\omega$ de la suite croissante de cardinaux transfinis $(\aleph_0, \aleph_1, \aleph_2, \ldots)$ ou, ce qui est équivalent, l'existence de l'ensemble $M = \{\aleph_i : i \in \mathbf{N}\}$. C'est pour combler cette lacune que Fraenkel a introduit en 1922 l'axiome de remplacement (ou de substitution).

Appelons *condition fonctionnelle* une phrase ouverte $F(t,x)$ représentant une fonction qui à tout t fait correspondre un et un seul x. Le schéma d'axiome de remplacement peut alors se formuler ainsi :

Pour tout ensemble a, si $F(t,x)$ est une condition fonctionnelle sur a, alors il existe un ensemble qui contient exactement ces éléments x pour lesquels $F(t,x)$ est vérifié par (au moins) un t membre de a. Ceci se note :

$$\forall z_1 \ldots \forall z_n \forall a \, [\forall u \forall v \forall w \, (u \in a \wedge F(u,v) \wedge F(u,w) \Rightarrow v = w) \\ \Rightarrow \exists y \forall x \, (x \in y \equiv \exists t \, (t \in a \wedge F(t,x)))] \,,$$

où u, v, w et y n'ont pas d'occurrences libres dans $F(t,x)$ et où $z_1, \ldots, z_n$ sont les variables libres de $F(t,x)$ autres que t et x.

On voit tout de suite qu'étant donné l'ensemble $\mathbf{N}$ et une phrase ouverte $F(t,x)$ exprimant une fonction f à arguments dans $\mathbf{N}$, on peut, grâce à l'axiome de remplacement, établir l'existence de l'ensemble $M = \{\aleph_0, \aleph_1, \aleph_2, \ldots\}$, ensemble qui regroupe les images de $\mathbf{N}$ par la fonction $f = \aleph$.

Une autre application de l'axiome de remplacement consiste à déduire, de l'existence (prévue par l'axiome de l'infini) de l'ensemble $\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \ldots\}$, l'existence de l'ensemble $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \ldots\}$ (si un élément est x, l'élément suivant est $x \cup \{x\}$). Ce deuxième ensemble infini est habituellement assimilé aux nombres naturels. En effet, dans le cadre de la théorie *ZF*, on *définira* les nombres naturels comme suit : $0 = \emptyset$, $1 = 0 \cup \{0\}$, $2 = 1 \cup \{1\}$, et ainsi de suite. L'avantage de cette modélisation est que l'ensemble représentant le naturel n

comporte exactement n éléments, qui sont les ensembles (correspondant à) $0, 1, \ldots, n-1$; la relation "$\in$" modélise ici la relation arithmétique "$<$".

1.6.9. *L'axiome du choix*

L'axiome du choix a été formulé par Zermelo. Il a tout de suite suscité des polémiques tenant au fait qu'il exigeait que s'opère une véritable mutation dans notre conception de la nature des ensembles. Un livre entier a été consacré par G.H. Moore en 1982 à l'étude des implications mathématiques et philosophiques de cet axiome. Pour désigner le système axiomatique de Zermelo-Fraenkel enrichi de l'axiome du choix, on utilise le sigle *ZFC*. Une version courante de l'axiome du choix est la suivante: "Si x est un ensemble composé d'ensembles non vides disjoints deux à deux (c'est-à-dire n'ayant pas de membres en commun), alors il existe au moins un ensemble y qui a exactement un élément en commun avec chacun des éléments de x". Si l'ensemble x est un ensemble fini, l'axiome est aisément démontrable à partir des axiomes de *ZF*. Il n'est non trivial que si l'ensemble x est infini. Dans ce cas, x peut être dénombrable (équipotent à l'ensemble des naturels) ou non dénombrable. Certains mathématiciens ne l'acceptent que pour x dénombrable.

Voici une formulation alternative de l'axiome du choix: "Pour tout ensemble t dans lequel l'ensemble vide n'est pas contenu, il existe une fonction de choix f dont le domaine d'arguments est t, telle que pour chaque s membre de t, $f(s)$ est membre de s". Il est essentiel ici de garder présente à l'esprit l'importante remarque d'Hadamard: "L'axiome pose l'*existence* d'une fonction de choix, pas sa *construction* (cité par G. Moore, 1982, p. 309)".

Bien que Cantor ne l'ait pas formulé, l'axiome du choix est présupposé dans plusieurs de ses démonstrations, comme le note Ph. Jourdain, et en particulier dans la démonstration du théorème "Chaque ensemble infini T a des parties qui ont le nombre cardinal $\aleph_0$ (Cantor 1895, 1955, p. 153)". L'axiome intervient aussi dans la démonstration de théorèmes de la théorie des ensembles aussi célèbres que contestés tels que la loi de trichotomie: "Pour tous ensembles a et b, a est d'une puissance supérieure à b ou égale ou inférieure". Cet énoncé n'est pas tautologique; au contraire, on peut fort bien concevoir que les puissances de deux ensembles infinis soient incomparables.

Moore a reconstitué la genèse de la démarche intellectuelle profondément originale codifiée dans l'axiome du choix par Zermelo, axiome constamment utilisé aujourd'hui dans les démonstrations en mathématiques. La première phase est le choix arbitraire d'un élément non spécifié. On la trouve présente dans les *Eléments* d'Euclide: "De tels choix étaient à la base de l'ancienne méthode où l'on prouvait une généralisation en considérant un objet arbitraire mais défini et en appliquant l'argument à cet objet (Moore 1982, p. 11)". L'ancêtre du choix codifié dans l'axiome du choix, sa forme la plus fruste, c'est donc la règle de logique qu'on appelle généralisation universelle. La

deuxième étape commence "quand un mathématicien fait une infinité de choix en énonçant une règle (*ibid.*)". La troisième étape est celle où un mathématicien fait une infinité de choix pour lesquels il pourrait fournir une règle, mais oublie ou s'abstient simplement de formuler la règle. C'est ce qui semble s'être passé dans la démonstration d'une des versions du théorème de la valeur intermédiaire donnée par Cauchy ("Toute fonction f sur les réels, continue dans l'intervalle fermé $[a, b]$, a une racine dans celui-ci pourvu que $f(a)$ et $f(b)$ soient de signes opposés."). La quatrième étape intervient quand on fait une infinité de choix et qu'aucune règle n'est possible. A ce moment l'axiome du choix est vraiment indispensable. Le premier exemple de preuve requérant l'axiome du choix serait une preuve écrite par Heine en 1872, démontrant un théorème d'analyse, preuve que Heine attribue d'ailleurs à Cantor.

Russell a donné un exemple extrêmement clair pour localiser la difficulté sur laquelle ont buté les mathématiciens et lui-même quand ils ont été pour la première fois confrontés à l'axiome du choix. Il imagine un ensemble de $\aleph_0$ paires de bottes et un ensemble de $\aleph_0$ paires de chaussettes. On veut prouver que le nombre de chaussures est pair et de même pour le nombre de chaussettes. Si on peut diviser chacune de ces deux classes en deux classes semblables, on aura établi que le nombre est pair dans les deux cas. Pour les chaussures, cette division ne fait pas problème car il y a une règle de choix à notre disposition pour diviser la classe en deux classes semblables: "nous n'avons qu'à mettre toutes les bottes droites dans une classe et toutes les bottes gauches dans l'autre: la classe des bottes droites est semblable à la classe des bottes gauches et notre problème est résolu (Russell 1906, 1986, p. 73)". Il n'en va pas de même pour les chaussettes: "Si le nombre de paires était fini, nous pourrions simplement choisir une de chaque paire; mais nous ne pouvons choisir l'une de chaque paire quand le nombre de paires est infini, à moins que nous n'ayons une règle de choix, et dans le cas présent une telle règle ne peut être trouvée (puisque les membres de chaque paire de chaussettes, contrairement à ceux de chaque paire de bottes, sont indiscernables) (Russell, *ibid.*)".

Dans le cas de l'axiome de Zermelo, écrit Russell, "... le doute quant à l'existence d'une *classe* qui fera la sélection découle du doute quant à l'existence d'une norme (Russell, *op. cit.*, p. 77)". Ce doute, cependant, ne peut guère être éprouvé par quelqu'un qui accepte l'axiome du choix. En effet, l'axiome du choix a précisément pour objectif de nous assurer que dans l'exemple considéré, il existe bien une classe (infinie) contenant une chaussette de chaque paire, même si les membres de cette classe ne sont pas rassemblables au moyen d'une norme. Notons que la classe infinie dont l'axiome du choix affirme l'existence n'est pas nécessairement unique. L'axiome du choix ne détermine pas univoquement l'ensemble x dont il établit l'existence, contrairement à l'axiome de remplacement qui opère avec une condition fonctionnelle.

Pour accepter dans toute sa force l'axiome de Zermelo, il faut rompre avec une idée reçue, la croyance que tous les ensembles sont les extensions

d'une phrase ouverte. Les antinomies, et tout particulièrement l'antinomie de Russell, permettent d'affirmer que toutes les phrases ouvertes ne déterminent pas un ensemble: aucun ensemble n'est spécifié par la phrase ouverte $x \notin x$, en d'autres termes, une phrase ouverte n'est pas une condition suffisante de l'existence d'un ensemble correspondant. Admettre l'axiome de Zermelo, c'est achever de dissocier la notion d'ensemble de celle de phrase ouverte, c'est reconnaître qu'une phrase ouverte n'est pas une condition nécessaire de l'existence d'un ensemble même infini.

Les logiciens ont eu peine à faire cette dissociation conceptuelle parce qu'ils avaient coutume de concevoir les ensembles comme l'extension de phrases ouvertes. Les ensembles finis hétéroclites dont on ne peut donner l'extension que par énumération ne font pas problème. Dans le cas des ensembles infinis, au contraire, nous sommes enclins à penser qu'on ne peut les appréhender qu'intensionnellement, au travers d'une propriété commune à tous leurs membres ou au travers d'une relation comme celle qui permet de définir les ensembles inductifs (ensembles qui contiennent l'ensemble vide et les successeurs de leurs membres). C'est pourquoi Russell, tout en montrant, par son exemple des ensembles de paires de chaussettes, qu'on a vraiment besoin de l'axiome du choix, partage les réticences du mathématicien Hobson qui estimait qu'un ensemble infini exigeait une norme (une phrase ouverte) et que des collections faites aussi au hasard que celles dont il est question dans l'axiome du choix sont en réalité des "non-entités".

Chez les mathématiciens, un autre obstacle ayant freiné l'acceptation de l'axiome du choix est l'adhésion tacite à une conception constructiviste de l'existence mathématique: n'existent que les objets mathématiques que nous pouvons construire à l'aide d'une norme ou d'une règle. Moore a bien mis ce point en lumière: "L'axiome du choix consacre, écrit-il, une transformation fondamentale qui se produisit à la fin du dix-neuvième siècle et au début du vingtième. Le recours à l'infini en acte un peu partout en mathématique, comme sous-produit de la théorie des ensembles, a rendu évident que les concepts d'existence et de construction sont des concepts mathématiques profondément différents. Comme Lebesgue et Hadamard l'ont tous deux reconnu très tôt, ce qui est en jeu dans la controverse sur l'axiome, ce sont deux conceptions diamétralement opposées des mathématiques. La volonté de restreindre les mathématiques exprimée par les constructivistes a acquis un nom (peut-être simplement parce qu'elle était la vue non courante) tandis que le point de vue épousé par Hadamard devenait la position généralement admise et ne trouvait pas d'autre nom que celui de mathématiques modernes (Moore 1982, pp. 309-310)".

1.6.10. *L'axiome de fondation*

Le schéma d'axiome de fondation affirme que s'il existe un élément x qui remplit la condition Fx, alors il existe un élément minimal u qui remplit la

condition Fx, ce qui se note formellement :

$$\exists x\, Fx \;\Rightarrow\; \exists u\,(Fu \wedge \forall y\,(y \in u \;\Rightarrow\; \neg Fu))\,.$$

Au sein d'un ensemble a peuvent exister des éléments x et y tels que le premier soit (aussi) un élément du second. Plus généralement, un ensemble peut contenir des suites finies ou non, dont chaque terme est élément du terme suivant. C'est le cas notamment de l'ensemble dont l'axiome de l'infini affirme l'existence : dans la suite (ensemble vide, ensemble ayant pour membre l'ensemble vide, ensemble ayant pour membre l'ensemble ayant pour membre l'ensemble vide, ...), chaque terme est élément du terme suivant. On voit donc que la relation d'appartenance peut déterminer, au sein d'un ensemble, une structure assez complexe.

Le schéma d'axiome de fondation restreint la complexité de cette structure, en interdisant l'existence de suites infinies $(x_0, x_1, x_2, \ldots)$ descendantes, c'est-à-dire telles que x_{i+1} soit élément de x_i, pour tout i.

D'une manière équivalente, l'axiome de fondation garantit l'existence, dans tout ensemble non vide a, d'un élément "minimal" b, ne contenant aucun élément de a. Cet énoncé très fort exige que toute suite descendante se termine après un nombre fini de pas. L'axiome de fondation a été adopté en 1930 par Zermelo. Pourquoi refuser l'existence aux ensembles non bien fondés ? Fraenkel, Bar-Hillel et Levy répondent "que nous ne pouvons invoquer d'autre argument en faveur de l'admission des ensembles non bien fondés parmi les ensembles qu'un seul argument : les tolérer satisfait notre désir de généralité (Fraenkel, Bar-Hillel et Levy 1973, p. 88)". Ils jugent cet argument de peu de poids "puisqu'aucun secteur de la théorie des ensembles ou des mathématiques n'a besoin d'ensembles non bien fondés". En d'autres termes, ils sont inutiles. Un autre argument a été avancé : on a dit que des ensembles qui admettraient des chaînes descendantes infinies ou des cycles d'appartenance (incluant l'auto-appartenance) seraient contre-intuitifs (Rogers 1971, p. 164).

1.7. Ensembles non bien fondés

En 1968, dans un article sur "les ensembles extraordinaires" publié par le *Bulletin de la Société mathématique* de Belgique, M. Boffa a démontré que l'on peut ajouter à ZF les six hypothèses "extraordinaires" suivantes qui nient l'axiome de fondation sans mettre en péril la consistance présumée de ZF :

$$\begin{aligned}
H1 &: \exists x\,(x = \{x\}) \\
H2 &: \exists x\,(x = \{0, x\}) \\
H3 &: \exists x \exists y\,(x = \{x\} \wedge y = \{y\} \wedge x \neq y) \\
H4 &: \exists x \exists y\,(x = \{x\} \wedge y = \{x, y\} \wedge x \neq y) \\
H5 &: \exists x \exists y\,(x = \{y\} \wedge y = \{x\} \wedge x \neq y) \\
H6 &: \exists x \exists y\,(x = \{y\} \wedge y = \{x, y\} \wedge x \neq y)
\end{aligned}$$

Ces hypothèses sont formalisables dans le langage de la logique du premier ordre avec identité. Par exemple l'hypothèse $H1$ peut être exprimée par

$\exists x \forall y (x \in y \equiv y = x)$. Elles sont vérifiées dans des structures dans lesquelles la relation d'appartenance a des propriétés inattendues. Par exemple, $H1$ est vérifiée par une structure où la relation R exprimée par le prédicat binaire d'appartenance est réflexive; $H5$ est vérifiée par une structure où R est symétrique.

Le mathématicien P. Aczel a formulé dans *Non-Well Founded Sets* (1987) une théorie des ensembles qui accueille les ensembles non bien fondés. Barwise et Etchemendy ont très clairement dégagé l'idée de base de la théorie d'Aczel : "La conception des ensembles d'Aczel procède directement de l'intuition qu'un ensemble est une collection de choses dont la relation (héréditaire) d'appartenance peut être représentée de manière non ambiguë par des graphes ... L'élément libérateur, c'est d'autoriser des graphes arbitraires, y compris des graphes qui contiennent des cycles ... Evidemment, des graphes avec cycles ne peuvent représenter les ensembles d'un univers bien fondé (Barwise et Etchemendy 1987, p. 37)".

La théorie des ensembles d'Aczel, comme celle de Boffa, est une extension de la version actuelle de *ZF*. (L'axiome de fondation a été découvert par Mirimanoff en 1917, introduit par von Neumann en 1925 et nommé ainsi par Zermelo en 1930.) Elle inclut les ensembles bien fondés de cette dernière, mais y ajoute les ensembles non bien fondés, appelés "hyper-ensembles" par certains. Ces hyper-ensembles se comportent, à bien des égards, comme les ensembles bien fondés. La théorie des ensembles où l'axiome de fondation est remplacé par un axiome d'antifondation n'est pas un jeu gratuit. Elle s'est révélée très féconde pour modéliser des théories récentes comme la théorie générale des situations de Barwise et Perry, mais aussi pour traiter mieux qu'on ne l'avait fait jusqu'ici certains paradoxes, notamment certaines formes particulièrement récalcitrantes du paradoxe du Menteur.

Rappelons l'une des formes les plus classiques du paradoxe du Menteur. Soit la phrase (Quine 1966, p. 9) :

(1) Cette phrase est fausse.

Nous devons admettre :

(2) La phrase "la neige est blanche" est vraie si et seulement si la neige est blanche.

Nous devons admettre aussi :

(3) La phrase "Cette phrase est fausse" est vraie si et seulement si cette phrase est fausse.

Or

(4) cette phrase = la phrase "cette phrase est fausse"

En remplaçant le sujet de "est vraie" dans (3) par son équivalent en vertu de la logique de l'identité, nous obtenons :

(5) Cette phrase est vraie si et seulement si cette phrase est fausse.

La proposition (5) ne peut être vraie, puisqu'elle a la forme $p \equiv \neg p$.

Exploitant la distinction due à Hilbert entre langue-objet et métalangue, Tarski (1934, 1972) a montré comment empêcher le paradoxe de se produire en introduisant des dénivellations qui n'existent pas dans la langue naturelle. Dans une langue formelle bien faite, le prédicat "vrai", qui s'applique à des phrases du langage-objet, ne peut être contenu dans ce langage. Il est refoulé dans le métalangage auquel appartiennent aussi les noms des expressions du langage-objet. On peut certes appliquer le prédicat "vrai" à une phrase du métalangage, mais ce prédicat appartient alors au métamétalangage. On arrive donc à toute une hiérarchie de langages.

En 1975, Kripke observe que la solution de la hiérarchie des langages était trop mutilante : elle rejette comme étant toujours paradoxales des phrases qui ne le sont qu'occasionnellement, en vertu de faits contingents. Il propose une nouvelle théorie de la vérité pour un langage qui évite aussi l'antinomie du menteur, qui tolère l'auto-référence (le prédicat "vrai" est réinterprété dans le langage objet) et qui possède son propre prédicat de vérité (Barwise et Etchemendy 1987, p. VIII).

En 1987, Barwise et Etchemendy reprennent à nouveaux frais le problème du paradoxe du Menteur en modélisant les "phénomènes de circularité" au moyen de la théorie des ensembles d'Aczel. J.P. Delahaye précise en ces termes leur apport : "La théorie de Tarski exclut sans expliquer. La théorie de Kripke résout le paradoxe du menteur mais pas le paradoxe du menteur renforcé (une version particulièrement sophistiquée de ce paradoxe)... La théorie austinienne de Barwise et Etchemendy résout tous les paradoxes préthéoriques et nous éclaire sur leurs causes profondes...".

Au crédit de la théorie des ensembles non bien fondés, on notera aussi qu'elle a servi récemment à fournir des modèles à la théorie axiomatique des propriétés. G. Cherchia et R. Turner (1988) estiment que les propriétés, contrairement aux ensembles ordinaires, ne sont pas bien fondées. Alors qu'on refuse qu'un ensemble ordinaire soit membre de lui-même, on désire qu'une propriété puisse être attribuée à elle-même. On veut, par exemple, rendre compte de la validité de l'inférence suivante (mentionnée par Parsons, 1979), "Tout a la propriété d'être identique à soi donc la propriété d'être identique à soi a la propriété d'être identique à soi", sans tomber dans le paradoxe de la prédication dont il sera question plus loin.

On pourrait penser que le paradoxe du Menteur prouve qu'il n'existe pas de langage universel, mais seulement une hiérarchie de langages, et que ce paradoxe est la réplique sémantique de l'antinomie établissant qu'il n'existe pas un ensemble de tous les ensembles. Ce serait accorder un poids indu à une analogie superficielle. Rivenc a souligné une importante différence entre les deux cas : "On peut faire remarquer qu'un tel 'ensemble' (l'ensemble de tous les ensembles) serait déjà bizarre indépendamment des antinomies puisque, se contenant lui-même, il ne correspondrait pas à l'idée naturelle qu'un ensemble est formé à partir d'éléments préexistants". Au contraire, "L'idée qu'il existe,

pour nous autres hommes, quelque chose comme le langage en tant que mode de rapport fondamental au monde à travers la variété des formes linguistiques (mode qui nous permet de référer à l'absent ou à l'abstrait, ou d'inférer selon des règles) est ... naturelle ... (Rivenc 1993, p. 12)". Que les métalangages ne traitent jamais que de fragments du langage ne prouve nullement qu'il n'existe pas de langage universel: "ce serait un bien curieux argument de prétendre conclure, du fait qu'une théorie exhaustive d'un certain objet est ou paraît impossible, à l'inexistence dudit objet (Rivenc 1993, p. 13)".

1.8. Antinomie et imprédicativité

L'ingrédient de base de la langue logique, depuis Frege, est la fonction propositionnelle, c'est-à-dire "quelque chose qui contient une variable, et exprime une proposition aussitôt qu'une valeur est assignée à x ... (Whitehead et Russell 1913, 1989, p. 272)"; "une fonction (propositionnelle) est ce qui dénote de manière ambiguë un certain élément d'une totalité, à savoir les valeurs de la fonction (*ibid.*, p. 273)". On serait tenté de dire qu'une fonction propositionnelle chez Russell est simplement ce que nous appelons aujourd'hui une phrase ouverte. Il faut se garder cependant d'identifier ces deux notions; ce serait commettre un anachronisme et une confusion. "Une fonction propositionnelle n'est ni un symbole linguistique, ni une entité ... mais un mode d'expression de la généralité (Rivenc 1993, p. 61)". Ainsi, par exemple, la fonction propositionnelle "x est humain" associe à l'argument Socrate la proposition "Socrate est humain".

Dans *On some Difficulties in the Theory of Transfinite Numbers and Order Types* (1906, 1986), Russell avait admis, à la lumière des antinomies, que certaines fonctions propositionnelles ne correspondent à aucune classe. Il les avait appelées fonctions propositionnelles "non prédicatives". Il suggère que la source des antinomies est à chercher dans "ce que nous pouvons appeler des processus ou des classes qui se reproduisent". "Il y a quelques propriétés, continue-t-il, telles que, étant donné n'importe quelle classe de termes ayant tous une telle propriété, nous pouvons toujours définir un nouveau terme ayant aussi la propriété en question. Ainsi nous ne pouvons jamais rassembler tous les termes ayant la dite propriété en un tout (Russell 1906, 1986, p. 61)".

Dans *Les mathématiques et la logique*, Poincaré franchit une étape décisive et propose une définition de l'imprédicativité qui précise l'origine des paradoxes: "... les définitions qui doivent être regardées comme non prédicatives sont celles qui contiennent un cercle vicieux (Poincaré 1906, 1986, p. 94)". Une définition de l'ensemble E qui introduirait la notion de l'ensemble E dans l'expression définissante contiendrait un tel cercle et serait donc non prédicative.

L'élimination des définitions imprédicatives écarte les paradoxes, pourvu que l'on prenne le mot "totalité" dans un sens assez large (Feferman 1987, p. 172), mais met en péril des définitions auxquelles on ne peut renoncer sans appauvrir

la mathématique classique. Il faut donc consentir soit à relâcher l'interdiction, soit à étendre la notion de prédicativité. Poincaré (1909) a choisi la seconde voie et proposé la définition suivante des concepts de prédicativité et de non-prédicativité : "Une définition d'un ensemble e qui utilise dans le définissant une totalité E dont e fait partie, est néanmoins prédicative, si e ne fait pas partie de E en vertu de sa définition, mais par suite d'une démonstration postérieure à la fois à la définition de E et à celle de e (cité par Heinzmann 1985, p. 62)". Dans cette définition, on aperçoit l'amorce d'une distinction très profonde entre la non-prédicativité vicieuse et la non-prédicativité inoffensive.

Ramsey a emprunté la première voie et disculpé l'imprédicativité. Il tient pour légitime le fait de décrire une proposition par référence à la totalité dont elle fait partie "de même que nous pouvons faire référence au plus grand homme dans un groupe, l'identifiant de la sorte au moyen d'une totalité dont il est lui-même membre sans qu'il y ait aucun cercle vicieux (Ramsey 1925, 1978, p. 192)".

Gödel a, comme Ramsey, pris la première voie et soutenu que les définitions imprédicatives ne doivent pas être rejetées en bloc. Il y a lieu de distinguer deux sortes de totalités, les totalités que nous construisons et celles qui existent indépendamment de nous. "S'il est question d'objets qui existent indépendamment de nos constructions, il n'y a rien d'absurde dans l'existence de totalités contenant des membres qui ne peuvent être décrits (être caractérisés univoquement) que par référence à cette totalité (Gödel 1944, 1990, p. 137)". Gödel cautionne donc certaine définitions imprédicatives. On peut, en effet, soutenir qu'il n'y a rien à redire au fait que l'entité X que l'on définit est définie en termes d'entités Y parmi lesquelles X figure.

Dans *Set theory and its logic*, Quine a réexaminé la question des responsabilités de l'imprédicativité dans la production des antinomies : "Et qu'en est-il maintenant du cercle vicieux ? Un raisonnement circulaire séduit sa victime en se concédant parmi les prémisses la thèse à démontrer, sans le savoir. Une définition circulaire introduit en contrebande le terme à définir dans le définissant. Mais la spécification imprédicative d'une classe ne fait aucune de ces deux choses (Quine 1963, p. 242)". Quine reconnaît qu'en condamnant la spécification imprédicative des classes, on se débarrasse de l'antinomie de Russell, mais ce procédé, pour efficace qu'il soit, ne peut prétendre démasquer une erreur cachée qui serait à l'origine des paradoxes : "La spécification imprédicative n'est pas plus manifestement vicieuse que le fait d'isoler un individu comme étant l'étudiant de Yale le plus typique en s'appuyant sur des moyennes relatives à une population qui le contient comme membre (*ibid.*, p. 243)". La spécification imprédicative d'une classe ne deviendrait vicieuse, aux yeux de Quine, que si l'on considérait que les classes sont créées au fur et à mesure qu'elles sont spécifiées au lieu d'être là dès le début.

1.9. La théorie des types de Russell

Dans *Les paradoxes de la logique*, Russell marque un accord partiel avec le diagnostic de Poincaré quant à l'origine des paradoxes: "Les paradoxes de la Logistique sont attribués par M. Poincaré à deux sources: des cercles vicieux, et la croyance à l'infini actuel. Nous sommes d'accord pour la première, mais non pour la seconde (Russell 1906, 1986, p. 142)". S'il est en partie d'accord avec Poincaré sur l'origine du mal, il ne l'est pas, comme le remarque D. Vernant, sur la thérapie: "... les cercles vicieux commis ont ceci de particulier qu'on ne peut pas les éviter simplement en observant qu'on les a commis; car l'assertion qu'ils doivent être évités (si elle n'est pas accompagnée d'une refonte des principes logiques) enveloppe elle-même un de ces cercles qu'elle prescrit d'éviter (Russell, *ibid.*, p. 142, cité par D. Vernant, 1993, p. 279)".

Supposons que nous admettions le mot d'ordre "Les cercles vicieux doivent être évités" et que nous cherchions à éviter le paradoxe des classes qui ne s'appartiennent pas à elles-mêmes en respectant ce mot d'ordre. A quoi ce dernier nous engage-t-il? Il nous engage à *restreindre les totalités.* Comme l'écrit F. Rivenc, "Quand je parle, disons, de toutes les classes qui ne s'appartiennent pas à elles-mêmes, cela doit s'entendre selon le principe du cercle vicieux, pas vraiment toutes, ou toutes non compris la classe paradoxale w qui présuppose cette totalité. Analogiquement, quand je parle de toutes les propositions, par exemple, pour énoncer la loi du Tiers exclu, cela doit s'entendre aussi non compris cette dernière proposition (la loi du Tiers exclu elle-même), puisqu'elle contient une variable apparente logée dans "toutes", qui se réfère à une totalité de propositions (Rivenc 1993, p. 102)".

Le problème surgit quand on veut transformer le mot d'ordre en théorie. Russell a clairement localisé celui-ci dans le passage suivant: "Si nous disons: 'Toutes les propositions sont vraies ou fausses, excepté la proposition que toute proposition est vraie ou fausse', nous n'évitons pas le cercle vicieux. Car c'est là un jugement portant sur toutes les propositions, à savoir: 'Toutes les propositions sont vraies ou fausses, ou identiques à la proposition que toutes les propositions sont vraies ou fausses'. Et cela suppose que nous connaissons le sens de 'Toutes les propositions sont vraies ou fausses', où *tout* ne comporte pas d'exception. Cela revient à définir la loi du Tiers exclu par: 'Toutes les propositions excepté la loi du Tiers exclu sont vraies ou fausses', où le cercle vicieux est flagrant (Russell 1906, 1986, p. 139)". Pour reprendre l'heureuse formulation de F. Rivenc, le problème réside dans le fait que "Toute théorie qui prétend dire les restrictions les transgresse dans l'acte de les dire (Rivenc, *ibid.*)".

Il y a deux solutions possibles: la première serait de faire appel à la distinction entre *montrer* et *dire* dont Wittgenstein fera usage dans le *Tractatus logico-philosophicus* (1921) et de "se résigner à simplement montrer les restrictions sans jamais formuler le principe du cercle vicieux lui-même", l'autre serait de "reconnaître la nécessité de quelque métathéorie antérieure

à la théorie logique elle-même, qui soit déliée des contraintes qu'elle pose, qui désobéisse aux lois logiques fondamentales dont elle affirme l'universelle validité (Rivenc, *ibid.*)".

Les deux solutions répugnent également à Russell qui professe *l'universalisme logique*, c'est-à-dire la doctrine affirmant qu'il y a une seule logique, logique qui est universelle en ce sens qu'elle s'applique à tout domaine d'objets pensables (Rivenc, *ibid.*). En 1908 cependant, Russell desserre les contraintes de l'universalisme logique et introduit la notion de *domaine de signification* pour les fonctions propositionnelles Φx, ce qui a pour effet de restreindre la variable x à la totalité des arguments possibles de ces dernières. La tâche — reconnue plus haut *impossible* — de formuler explicitement la restriction est devenue *superflue*: "... si la fonction cesse d'avoir une signification dès que la variable sort d'un certain domaine, la variable est alors *ipso facto* confinée à ce domaine, sans qu'il soit besoin de le formuler explicitement (Russell 1908, trad. Rivenc 1992, p. 322)".

Le principe du cercle vicieux sous sa forme courante ("ce qui contient une variable liée ne peut être une valeur de cette variable") permet d'écarter la classe des classes qui ne se contiennent pas elles-mêmes mais "... à lui seul, il ne paraît pas interdire la construction de la fonction 'ne pas être prédicable de soi-même', non plus que l'application de cette fonction à elle-même (forme intensionnelle du paradoxe de Russell) puisqu'aucune quantification n'est à l'œuvre dans cette fonction (Rivenc 1993, pp. 128-129)".

Pour enrayer le paradoxe de l'imprédicable, il faut poser un nouveau principe "qui dit qu'une fonction présuppose la totalité de ses valeurs et donc de ses arguments (Rivenc, 1993, p. 129)". Or, ce principe, joint au principe du cercle vicieux (selon lequel les valeurs d'une fonction ne peuvent contenir de termes définissables seulement à partir de la fonction) interdit qu'une fonction se prenne comme argument. La fonction Φ ne peut donc prendre pour argument Φ, ou quelque chose qui contient Φ. En d'autres termes, $\Phi(\Phi)$, ou *Imprédicable*(*Imprédicable*), n'exprime pas une proposition.

La contrainte sur les fonctions propositionnelles mentionnée plus haut bloque l'antinomie de l'imprédicable, mais elle n'est pas directement applicable aux paradoxes mathématiques. Elle le devient cependant si l'on traite, comme le fait Russell, la classe comme un objet dérivé: "Pour résoudre la contradiction portant sur la classe des classes qui ne sont pas membres d'elles-mêmes, nous supposerons ... qu'une proposition portant sur une classe doit toujours être réduite à un énoncé portant sur une fonction qui définit cette classe, c'est-à-dire sur une fonction qui est satisfaite par tous les membres de la classe et par aucun autre argument ... Aussi une classe ne peut-elle, en vertu du principe du cercle vicieux, être l'argument de la fonction qui la définit sans que le tout soit privé de signification, c'est-à-dire que, si nous dénotons par $\{x, \Phi x\}$ la classe définie par Φx, le symbole $\Phi\{x, \Phi x\}$ doit être dépourvu de sens (Whitehead et Russell 1910, 1989, p. 304)".

Russell conserve la terminologie de Poincaré: il appelle prédicatives les fonctions propositionnelles qui n'enveloppent pas de cercle vicieux et se donne pour tâche de formuler des règles générales de construction des fonctions qui nous assureront qu'une fonction propositionnelle de complexité quelconque est prédicative. Dans le cas de fonctions propositionnelles élémentaires, on dira que toute fonction dont les arguments sont des variables d'individu est prédicative. Dans le cas des fonctions de fonctions d'individu, on imposera comme seuls arguments permis les fonctions d'individu. Par exemple "est génial" pourra se dire de Cantor et "est rare" pourra se dire de "être génial". Ce procédé de construction peut être généralisé: on obtient une hiérarchisation des fonctions telle que chaque fonction devienne condition de déterminabilité d'une fonction immédiatement supérieure. La règle est "qu'une fonction ne peut avoir pour type ... que des arguments d'un type immédiatement inférieur au sien (Vernant 1993, p. 291)", où par "type" il faut entendre le domaine des valeurs possibles de la variable, c'est-à-dire le domaine de signifiance de la fonction.

Cette version simple de la théorie des types hiérarchise les fonctions sur base du type des variables libres. Elle permet d'écarter certaines contradictions, mais pas toutes. Elle ne dit rien des fonctions propositionnelles complexes dans lesquelles figurent, par exemple, des variables liées, comme c'est le cas dans "x a toutes les qualités d'un grand général". Si on ne dispose que de la théorie simple des types, on ne fera pas de distinction entre la fonction propositionnelle précitée et des fonctions comme "x est intelligent", "x est courageux", et on commettra le cercle vicieux dénoncé par Poincaré.

C'est pour respecter ce principe de Poincaré qui interdit que si la définition d'un objet requiert de quantifier sur une totalité, l'objet en question soit membre de cette totalité, que Russell crée la théorie ramifiée des types qui instaure "une nouvelle hiérarchisation à l'intérieur de la première, ayant pour objet de scinder chaque type de fonctions en différents ordres exclusifs (Vernant, *ibid.*, p. 292)". Bien que la propriété exprimée par la fonction propositionnelle "x a toutes les qualités qui font un grand général" soit elle-même une propriété d'un grand général, elle ne peut être l'une des qualités mentionnées dans la définition de la fonction propositionnelle en question. La théorie ramifiée des types assigne à cette fonction propositionnelle le rang de fonction propositionnelle du deuxième ordre alors qu'elle assigne à "x est intelligent", "x est courageux", etc., le rang de fonction propositionnelle du premier ordre.

Une fois en possession de la notion d'ordre, Russell donne un contenu technique positif à la notion de prédicativité de Poincaré: "Une fonction à une variable d'un ordre immédiatement supérieur à l'ordre de son argument sera appelée fonction prédicative; et le même nom sera donné à une fonction à plusieurs variables lorsque, parmi ces variables, il y en a une telle que la fonction devient prédicative par rapport à elle quand des valeurs sont assignées à toutes les autres variables (Russell 1908, 1992, p. 327)".

La théorie des types ne nous oblige pas à rejeter comme mal formé l'énoncé : "Tout homme normal possède quelques traits anormaux", mais elle nous impose, en revanche, de reconnaître que le prédicat "normal" figurant deux fois dans cet énoncé est ambigu et qu'il doit s'exprimer dans une langue bien faite par une fonction propositionnelle du premier ordre pour ce qui est de sa première occurrence dans l'énoncé, et par une fonction propositionnelle du deuxième ordre pour ce qui est de la deuxième occurrence.

La hiérarchisation des fonctions propositionnelles en ordres introduite par la théorie des types s'étend tout naturellement aux propositions et permet de désamorcer le célèbre paradoxe du Menteur proféré par Epiménide le Crétois. Après avoir formulé le point en ces termes : "La forme la plus simple de la contradiction nous est offerte par l'homme qui dit 'je mens'; s'il ment, il dit la vérité, et vice versa (Russell 1910, 1989, p. 301)", Russell en appelle à la dénivellation des ordres pour désamorcer le paradoxe : "... le mot 'faux' est ambigu, et ... pour lui ôter son ambiguïté, nous devons spécifier l'ordre de la fausseté, ou ce qui revient au même, l'ordre de la proposition à laquelle la fausseté est assignée ... si p est une proposition du $n$ième ordre, une proposition dans laquelle p figure comme variable apparente n'est pas du $n$ième ordre, mais d'un ordre plus élevé. Donc l'espèce de vérité et de fausseté qui peuvent appartenir à l'énoncé 'il y a une proposition p que j'affirme et qui a la fausseté du $n$ième ordre' sont une vérité ou une fausseté d'un ordre plus élevé que le $n$ième. Aussi l'énoncé d'Epiménide ne porte-t-il pas sur lui-même et par conséquent ne surgit aucune contradiction (Whitehead et Russell, *ibid.*, pp. 303-304)".

La théorie des types nous oblige à remplacer les expressions "toutes les propositions", "toutes les propriétés" par "toutes les propositions d'ordre n", "toutes les propriétés d'ordre n". Ce faisant, elle écarte les totalités illégitimes qui conduisaient à des contradictions, mais elle entraîne conjointement certains effets indésirables. Nous en mentionnerons deux.

Premièrement, on ne peut plus formuler dans toute sa généralité le principe d'induction mathématique qui affirme que toute propriété possédée par 0 et par tout successeur d'un nombre naturel qui la possède est possédée par tout nombre naturel. Deuxièmement, on doit renoncer à définir la notion de borne supérieure en termes de coupure des rationnels.

Rappelons la définition russellienne de "nombre réel" et celle de "borne supérieure". Pour Russell, un nombre réel est une coupure inférieure des rationnels et une coupure inférieure des rationnels est une partie propre non vide de l'ensemble des rationnels, qui ne peut contenir le rationnel x sans contenir tout rationnel inférieur à x. Dans ce contexte, la borne supérieure d'un ensemble de nombre réels est simplement la réunion de ces nombres, à condition que cette réunion reste une partie propre de l'ensemble des rationnels.

Nous avons vu plus haut que selon les *Principia mathematica*, toute classe a sa caractéristique définissante. En vertu des définitions mentionnées plus haut, on peut dire qu'un nombre réel a une caractéristique définissante, qui

est une fonction des rationnels. Russell est dès lors confronté à une difficulté que J. Pieters décrit en ces termes : "en vertu de la théorie ramifiée des types, il convient que toute caractéristique définissante, par cela même qu'elle est une fonction propositionnelle, soit limitée à un ordre donné. Cet ordre est fini, étant donné que le nombre des arguments et des variables apparentes dans une fonction est lui-même fini. Appelons n l'ordre le plus élevé que l'on rencontre parmi les caractéristiques qui définissent les réels. Considérons alors un ensemble E de réels dont les caractéristiques définissantes sont de l'ordre n. En vertu de ce qui précède, la borne supérieure de l'ensemble E pourra être analysée comme la classe de tous les rationnels qui satisfont la fonction de satisfaire une quelconque des fonctions d'ordre n qui déterminent les membres de E. Or, en vertu de la hiérarchie des ordres, la fonction de satisfaire une quelconque fonction d'ordre n, n'est pas elle-même d'ordre n, mais d'ordre $n+1$. La caractéristique définissante de la borne supérieure de l'ensemble E est donc d'un ordre supérieur à celui des nombres réels. D'où il faut conclure que la borne supérieure de l'ensemble E n'est pas elle-même un nombre réel (Pieters 1981, p. 51)".

Pour résoudre ces problèmes, Russell énonce l'axiome de réductibilité : "Toute fonction propositionnelle est équivalente, pour toutes ses valeurs, à une certaine fonction prédicative (Russell 1908, 1982, p. 331)". Grâce à cet axiome, il est possible de réduire l'ordre des fonctions propositionnelles non prédicatives et en particulier, dans le cas de la définition de la borne supérieure, de ramener l'ordre $n+1$ de la caractéristique définissante de la borne supérieure de l'ensemble E à l'ordre n qui est l'ordre des réels.

La théorie pure des types permet d'écarter les antinomies purement logiques telles que celle du plus grand cardinal ou celle des classes qui ne sont pas membres d'elles-mêmes. Ramsey caractérise ainsi ces antinomies : "Elles n'enveloppent aucune autre idée que celles de classe, de relation et de nombre, elles peuvent être formulées dans le symbolisme logique et font leur apparition dans le développement effectif des mathématiques quand celui-ci se fait dans la bonne direction (Ramsey 1926, 1978, p. 227)".

La théorie ramifiée des types permettait, en outre, d'écarter les antinomies qu'on appelle aujourd'hui "sémantiques" et dont Ramsey dit qu'"elles n'apparaissent pas dans les mathématiques, mais dans nos considérations à propos des mathématiques; en sorte qu'il se peut qu'elles ne proviennent pas d'une faute de logique ou de mathématique, mais d'ambiguïté dans les notions psychologiques ou épistémologiques de signification ou d'assertion (Ramsey, *ibid.*, p. 228)".

Si l'on fait appel à l'axiome de réductibilité qui annule certains effets de la théorie ramifiée des types, "on peut craindre le danger, comme l'écrit Church, que (cet axiome) ne restaure les antinomies sémantiques que l'on cherchait à éviter au moyen de la théorie ramifiée des types (Church 1977, *in* Martin 1984, p. 304)". Cette crainte n'est pas fondée. Nous rappelons dans ses grandes lignes les raisons que donne Church pour la dissiper.

On se donne L, le langage de la théorie ramifiée des types. Au sein de L, on définit des sous-langages $L_1, L_2, L_3, \ldots$ tels que L_n est le sous-langage obtenu en effaçant toutes les variables et constantes d'ordre plus grand que n et en autorisant les variables d'ordre n à figurer seulement comme variables libres. Ces sous-langages forment une hiérarchie dans laquelle L_{n+1} peut être regardé comme un métalangage sémantique de $L_n \ldots$. Il reste alors à inclure les axiomes de réductibilité dans la hiérarchie des langages $L_1, L_2, L_3, \ldots$ chacun à sa place. Cela fait, "il semble donc que la hiérarchie de langages résultante continuera à se conformer à la (méthode) de Tarski pour résoudre les antinomies sémantiques (Church, *ibid.*)".

La théorie des types a été formulée pour un calcul des prédicats comportant des variables d'individus, des variables de fonctions propositionnelles du premier ordre, des variables de fonctions propositionnelles du deuxième ordre, etc. Pour faciliter la comparaison avec la théorie des ensembles *ZF*, nous la reformulerons pour un langage ensembliste.

La théorie des ensembles bloquait le paradoxe de Russell sur les ensembles qui ne se contiennent pas eux-mêmes en restreignant l'axiome de compréhension, c'est-à-dire en remplaçant cet axiome par l'axiome de séparation. La théorie des types ne touche pas aux axiomes de la théorie des ensembles; elle ne modifie pas l'axiome de compréhension, mais modifie le langage de la théorie. Elle classe les variables en types et exige que le type de la variable à droite du signe d'appartenance soit supérieur d'une unité au type de la variable à gauche du signe d'appartenance. Toute infraction à cette règle engendre, selon Russell, le non-sens.

Formulé dans le langage de la théorie des ensembles, l'axiome de compréhension non restreint qui permettait d'engendrer le paradoxe des ensembles qui ne se contiennent pas eux-mêmes s'écrit (l'indice représentant le type des variables) :

$$\exists\, y_{i+1}\, \forall\, x_i\, (x_i \in y_{i+1} \;\equiv\; F x_i)\,.$$

La substitution de la fonction propositionnelle $x_i \notin x_i$ à $F x_i$ est interdite pour deux raisons : d'abord parce qu'elle est non prédicative (l'instantiation de x en y revient à inclure une classe y dans le domaine des valeurs d'une quantification qui servait à la spécifier), ensuite parce qu'elle viole la règle des types, la variable à droite du symbole d'appartenance étant la même (donc de même type) que celle qui est à gauche.

1.10. Théorie des types et théorie des ensembles

Si nous adoptons la théorie des types, chaque niveau dans la hiérarchie des types a sa propre classe universelle, sa propre classe vide, ses propres nombres cardinaux. Dans la phrase "Il y a neuf planètes et neuf constellations", le mot "neuf" est employé de manière équivoque : il désigne dans sa première occurrence une classe de classes et dans sa deuxième occurrence une classe de

classes de classes. Comme l'écrivent Fraenkel, Bar-Hillel et Levy, "beaucoup de mathématiciens jugeront intolérable cette duplication, non seulement pour des raisons de simplicité, mais aussi et principalement parce qu'elle limite sévèrement leur liberté coutumière d'expression et de notation et qu'elle impose souvent des opérations techniques compliquées pour surmonter les restrictions de notation non désirées (Fraenkel, Bar-Hillel et Levy 1973, p. 160)".

La théorie des ensembles *ZF* est formulée dans le calcul des prédicats du premier ordre. Il n'y a donc qu'un seul univers du discours, un seul type de variables. Dans la théorie des types, au contraire, on se trouve en face d'une hiérarchie d'univers du discours ou de domaines de variables. Il y a toujours moyen de remplacer les quantificateurs restreints par des quantificateurs généralisés. Par exemple, les écritures

$$(\forall x \in \mathbf{N})\, Fx \quad \text{et} \quad (\exists x \in \mathbf{N})\, Fx$$

apparaissent respectivement comme des variantes syntaxiques de

$$\forall x\, (x \in \mathbf{N} \;\Rightarrow\; Fx) \quad \text{et} \quad \exists x\, (x \in \mathbf{N} \wedge Fx)\,.$$

Dans *Set Theory and its Logic*, Quine présente une théorie des types dans une logique non typée qui opère avec un domaine de variables unique. Les formules

$$\forall x_n\, Fx_n \quad \text{et} \quad \exists x_n\, Fx_n$$

seront remplacées respectivement par

$$\forall x\, (T_n x \;\Rightarrow\; Fx) \quad \text{et} \quad \exists x\, (T_n x \wedge Fx)\,.$$

Moyennant l'adjonction d'un axiome, tous les théorèmes de la théorie des types standard sont préservés. Comme dans la théorie des types, il y avait deux protections contre les paradoxes (la limitation du champ de variation et la dénivellation entre les types), on peut sans crainte se passer de la deuxième restriction, qui est superflue, et se contenter de la première.

Deux raisons subsistent, cependant, de préférer la théorie *ZF* à cette version améliorée de la théorie des types. Premièrement, le problème posé par la duplication gratuite des nombres n'est pas entièrement résolu : "Le maintien des indices de types (évité pour les variables) continue à être nécessaire en pratique pour toutes les constantes, en dépit de notre réduction de la notation de base à des variables générales (Quine 1963, p. 283)". Deuxièmement — et cette objection nous paraît plus grave — la contrainte de prédicativité telle qu'elle a été formulée empêche de démontrer le théorème de Cantor qui dit que toute classe a plus de sous-ensembles que de membres (Quine, *ibid.*, p. 265). Or ce théorème est une des pièces maîtresses de la théorie des ensembles.

L'examen de la démonstration d'un autre théorème célèbre de Cantor, selon lequel l'ensemble des nombres réels compris entre 0 et 1 n'est pas dénombrable, permet de bien voir à quel moment on doit violer la contrainte de prédicativité.

Dans la démonstration indirecte, on suppose que les nombres réels dans l'intervalle [0,1] sont dénombrables. Chacun de ces nombres réels peut s'écrire

sous la forme d'une fraction décimale illimitée, par exemple (Breuer 1961, p. 33):

$$1/2 = 0,499999\ldots \; \pi/6 = 0,523598\ldots$$

Si les nombres réels de l'intervalle en question sont dénombrables, ils peuvent être rangés dans une suite dénombrable de fractions décimales $A_1, A_2, A_3, \ldots$

Ces fractions décimales forment un "carré infini" dont voici une représentation partielle:

$$\begin{array}{lccccc} A_1: & 0, & z_{11} & z_{12} & z_{13} & \ldots \\ A_2: & 0, & z_{21} & z_{22} & z_{23} & \ldots \\ A_3: & 0, & z_{31} & z_{32} & z_{33} & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \end{array}$$

Le symbole z_{ik} est le $k$ième chiffre après la virgule de la fraction décimale représentant le nombre réel A_i. Une *fraction diagonale* relative à ce carré est un réel N dont le développement décimal $0, a_1 a_2 a_3 \ldots$ est tel que a_i diffère de z_{ii}, pour tout i.

Cette fraction N peut être formée d'une infinité de façons; elle représente un réel compris entre 0 et 1 mais elle diffère de A_n (au moins par sa $n$ième décimale) pour chaque $n \in \mathbf{N}$.

G. Heinzmann observe que "la démonstration est non prédicative parce qu'elle utilise dans la définition de N l'ensemble $[0,1]$ sous forme d'un quantificateur universel sur les A_n (Heinzmann 1985, p. 40) mais, ajoute-t-il, cette définition n'est pas suffisamment éclairante pour la question de la prédicativité". Il existe en effet une démonstration modifiée — démonstration directe dans sa partie centrale — qui prouve que la méthode diagonale n'est pas elle-même responsable de la non-prédicativité: "... prenons une collection M de nombres réels quelconques mais dénombrables, énumérés par une suite A_n, $n \in \mathbf{N}$. On a

$$\forall X \, (X \in M \equiv \text{“}X \text{ est énuméré par un } A_n\text{”}).$$

On définit ensuite le nombre diagonal N par rapport à M, mais N ne fait pas partie de M. Le moment non prédicatif ne survient que si nous passons de l'énoncé positif, 'pour chaque ensemble dénombrable de nombres réels il existe des nombres réels non contenus dans cet ensemble', à sa forme négative 'les nombres réels sont non dénombrables' (Heinzmann 1985, pp. 40-41)".

Il est cependant possible de libéraliser la théorie des types de manière telle qu'elle permette de reformuler en son sein les axiomes de la théorie des ensembles de Zermelo et de rendre indiscernables les deux théories rivales. La première forme que prend cette libéralisation consiste à rendre cumulative la théorie des types: chaque type inclut tous les types inférieurs à lui. Cette libéralisation dont Chwistek (1924) a été l'initiateur a permis à Quine (1963, pp. 272-279) de reconstruire la théorie des ensembles à l'intérieur de la théorie des types. Dans la version cumulative de la théorie des types élaborée par Quine, l'antinomie de Russell est évitée non par les contraintes syntaxiques

de la théorie des types elle-même mais grâce à l'axiome de séparation. Par ailleurs le célèbre théorème de Cantor concernant la taille de l'ensemble des parties d'un ensemble redevient démontrable.

Faisant le trajet inverse, D. Scott a présenté une axiomatisation de *ZF* inspirée par la théorie libéralisée des types (Scott 1974). Le système axiomatique de Scott est formulé dans un langage du premier ordre où $\in$ et le concept de niveau sont pris comme termes primitifs. J.W. Degen a construit un système de logique cumulative (K^*_Δ) dans lequel $\in$ et le concept de quasi-univers (qui correspond au concept de niveau chez Scott) sont définis (Degen 1983).

Une autre libéralisation prolonge la première, à savoir le passage de la théorie finie des types à la théorie transfinie (Andrews 1965).

Cette double libéralisation confère à la théorie des types la puissance de la théorie des ensembles classique, mais il y a un prix à payer : elle oblige à sacrifier l'esprit de la théorie russellienne des types. En effet, cette dernière, s'inspirant de Frege (Imbert 1969, Largeault 1971), hiérarchisait des concepts plutôt que des objets (ensembles); cela la rendait peut-être moins utile en mathématique, mais plus utile en logique pure (Cf. Smiley, Introd. à Ramsey 1978 et, sur la spécificité de la logique, de Rouilhan 1988).

Signalons pourtant que, comme fondement des mathématiques, la théorie des ensembles *ZF* (ou l'une de ses proches variantes) est aujourd'hui généralement acceptée: "... les solutions alternatives proposées à la théorie des ensembles standard n'ont pas réussi à s'imposer et ce rejet de fondements de rechange semble reposer sur de bonnes raisons (Urquhart 1989, p. 90)".

1.11. La métathéorie de la théorie des ensembles

Le théorème de Gödel de 1931 ayant établi que tout système assez riche pour contenir l'arithmétique ne peut être prouvé consistant de l'intérieur, c'est-à-dire avec les ressources du système, la seule démarche qui puisse être envisagée pour la théorie axiomatique des ensembles est de chercher une preuve de consistance relative.

Pour prouver la non-contradiction d'une théorie T, on construit un modèle de la théorie T dans une théorie T' déjà fondée. Ce type de démonstration de consistance relative est inapplicable en bloc à la théorie des ensembles en raison de sa position de pierre angulaire de la mathématique, position que lui reconnaît Zermelo: "La théorie des ensembles est cette branche des mathématiques qui a pour tâche d'étudier mathématiquement les concepts fondamentaux de nombre, d'ordre et de fonction dans leur simplicité originelle et, par la même, de développer les fondements logiques de l'arithmétique et de l'analyse tout entières; elle constitue de la sorte une composante indispensable de la science mathématique (Zermelo 1908, 1992, p. 370)". Si une démonstration de la consistance relative est impossible, en revanche des preuves ponctuelles de consistance relative ont été réalisées, montrant que si les axiomes A_1 à A_n

forment un ensemble consistant, l'adjonction de l'axiome A_{n+1} préserve la consistance initiale. Citons à titre d'exemple la preuve de non-contradiction (ou consistance) relative de l'axiome de fondation, dans *La théorie axiomatique des ensembles* de J.L. Krivine (pp. 58-59).

Une autre question que l'on peut se poser est celle de savoir si les axiomes sont indépendants les uns des autres. Si ce n'est pas le cas, si un axiome découle des autres, il est superflu. On a fait passer pour axiome ce qui est en fait un théorème. Pour démontrer que l'axiome A n'est pas déductible des axiomes B_1 à B_n, on construit un modèle qui vérifie B_1 à B_n et qui falsifie A.

Si parmi les axiomes, il y en a un qui est moins évident que les autres, il est particulièrement souhaitable de pouvoir montrer qu'il est déductible des autres. Il y a un axiome dans *ZFC*, à savoir l'axiome du choix, qui a suscité beaucoup de controverses et que l'on aurait voulu pouvoir soit démontrer soit réfuter en partant des autres axiomes. En 1906 Russell envisageait comme possible une réfutation de cet axiome au moyen d'une *reductio ad absurdum*. Il n'excluait pas non plus qu'on parvienne à le démontrer, mais jugeait cette éventualité beaucoup moins probable. Poincaré jugeait cet espoir illusoire. Les recherches ultérieures ont donné raison à Poincaré.

En 1938, Kurt Gödel a démontré que l'axiome du choix était logiquement compatible avec les autres axiomes de la théorie des ensembles; en 1963, Paul Cohen a démontré que l'axiome du choix ne pouvait être déduit des autres axiomes. L'axiome du choix est donc indépendant de la théorie *ZF*, de même que le célèbre "postulat des parallèles" est indépendant des autres axiomes de la géométrie euclidienne.

En combinant les résultats de Gödel et de Cohen, on arrive à une autre conclusion métathéorique importante : l'indépendance de l'hypothèse du continu. L'hypothèse du continu, c'est l'affirmation qu'il n'y a pas de puissance intermédiaire entre la puissance de l'ensemble des naturels ($\aleph_0$) et la puissance des réels, qui s'identifie donc à $\aleph_1$. Si nous acceptons le principe de bivalence, nous serons tentés de dire que l'une des deux propositions (l'hypothèse du continu ou sa négation) est vraie, mais que nous ne saurons jamais laquelle est vraie. A moins de dire que l'on pourra trancher en complétant *ZF* par un nouvel axiome. Mais pour que cela soit recevable, il faut que cet axiome ne soit pas *ad hoc*.

Quels sont les arguments que l'on peut avancer en faveur de nouveaux axiomes en théorie des ensembles ? Ce problème est central pour l'épistémologie des mathématiques; il n'est pas purement académique, puisque maints nouveaux axiomes ont été proposés au cours de ces dernières décennies. Penelope Maddy a montré dans *Believing the axioms* le "rôle des arguments non démonstratifs en théorie des ensembles, spécialement dans la recherche de nouveaux axiomes (Maddy 1988, p. 758)". Elle en recense huit qui sont les analogues de formes de confirmation ayant cours dans les sciences de la nature. Ceci pose la question des rapports de la théorie des ensembles avec le monde

physique. P. Maddy situe ces rapports dans le fait que "Les deux piliers de la théorie des ensembles : la théorie des nombres et la géométrie-analyse sont des disciplines qui se prêtent aux applications. Tandis que la théorie des nombres trouve sa source dans l'action de compter, la géométrie procède de l'étude des formes des objets ... et l'analyse de l'étude de leurs mouvements. La théorie des ensembles systématise et explique ces deux branches ... et doit à ce rôle une bonne part de sa crédibilité (*ibid.*, p. 761)". Elle rappelle aussi que le problème du continu est lui-même une question à propos des nombres réels de la physique.

Sur ces questions philosophiques, nous renvoyons le lecteur à l'article de J.P. Delahaye *Le réalisme en mathématiques et en physique* (1991).

Chapitre 2
Théorie des ensembles

2.1. Introduction

Plusieurs raisons confèrent à la théorie des ensembles une importance et un statut particuliers. Tout d'abord, elle passe pour constituer la base des mathématiques. Une telle affirmation est sujette à caution mais on observe que la plupart des développements mathématiques font appel à la théorie des ensembles, explicitement ou non; en particulier, l'analyse mathématique, la topologie et une bonne partie de l'algèbre reposent, d'une manière assez directe, sur cette théorie. Même les exposés de logique élémentaire font usage du vocabulaire ensembliste.

Le fait que tous les concepts mathématiques usuels (nombres, suites, objets géométriques, fonctions, opérateurs, etc.) puissent être étudiés dans le cadre de la théorie des ensembles n'a pas reçu de justification théorique vraiment satisfaisante. Certains auteurs, d'ailleurs, se sont intéressés à d'autres théories (catégories, matroïdes, etc.) susceptibles, elles aussi, de "fonder les mathématiques".

Signalons enfin que le rôle fondateur de la théorie des ensembles consiste simplement à décrire les divers concepts mathématiques dans un langage unique, formalisé et généralement commode. Il serait abusif, et surtout peu productif, de vouloir "enfermer" les mathématiques dans un cadre formel rigide, quelles que soient ses qualités.

Développer méthodiquement la théorie des ensembles a aussi l'intérêt de mettre en évidence l'utilité de la logique formelle en mathématique. En effet, la notion d'ensemble étant très intuitive, on pourrait penser que sa formalisation n'apporte rien; pourtant, comme on le verra, l'intuition ne suffit pas à rendre

compte de la richesse de cette notion. Dans la théorie des ensembles, idées intuitives et développements formels concourent à l'obtention de résultats d'un grand intérêt. Signalons au passage que la distinction entre "théorie naïve" et "théorie axiomatique" n'est pas une opposition. Simplement, l'étude du concept d'ensemble peut être menée plus ou moins loin, et avec plus ou moins de rigueur et de formalisme.

Ce qui est vrai en mathématique reste vrai en informatique : la formalisation des objets et surtout des raisonnements faits à leur propos est cruciale dans de nombreuses branches de l'informatique et, en particulier, dans la théorie des programmes. En fait, les programmes sont des objets mathématiques, puisqu'ils sont écrits dans un langage formel. Une réalité pragmatique rend la rigueur formelle spécialement utile dans l'étude des programmes : contrairement à la plupart des objets mathématiques, les programmes ne peuvent pas être décrits brièvement; or le risque d'erreur croît avec la longueur des textes ! Aborder la théorie des ensembles permet aussi d'illustrer les problèmes rencontrés lors de la modélisation d'une certaine réalité un peu floue. Ce type de démarche est crucial pour l'informaticien, puisque toute activité de programmation implique une activité préalable de modélisation (explicite ou non).

La théorie des ensembles permet de *construire* des objets mathématiques nouveaux; c'est ce qui la distingue des théories mathématiques usuelles, dans lesquelles les objets sont simplement *définis*, en langage naturel le plus souvent. Par contre, cet aspect rapproche la théorie des ensembles de l'informatique puisque programmer, c'est construire un nouvel objet (le programme). Dans le cadre de la théorie des ensembles, la première étape consiste à reconstruire des objets déjà bien connus, comme les nombres et les figures géométriques. Cette démarche peut, de prime abord, paraître inutile. Pourtant, de nombreuses difficultés rencontrées lors de l'étude des objets "simples" ont pu être résolues après que ces objets aient été correctement modélisés; l'utilité de la démarche (re)constructive s'en est trouvée démontrée. Néanmoins, des objets aussi élémentaires que les nombres font à ce point partie de notre univers que toute tentative de les construire à partir d'autres objets paraît artificielle. Nous montrerons que cette impression est fausse; bien au contraire, une telle construction repose directement sur les propriétés les plus intuitives et les plus utiles des objets étudiés. L'informaticien s'inspire directement des constructions ensemblistes pour définir des types de données.

La motivation initiale de la théorie des ensembles était l'existence d'une série de phénomènes troublants, liés de près ou de loin à la notion d'infini.[1] Cette

[1] Par exemple, l'énoncé "le tout est plus grand que la partie" ne s'applique pas aux ensembles infinis, comme nous le verrons plus loin. Plus généralement, une grande variété de propriétés ensemblistes, habituellement considérées comme évidentes, deviennent douteuses ou même absurdes si on les applique à des ensembles infinis.

motivation s'est élargie mais reste valable aujourd'hui. Plusieurs phénomènes de ce type seront évoqués. Le concept d'infini est important en logique élémentaire, comme le montrent le théorème de compacité et le théorème de Löwenheim et Skolem (§I.9.4.1). Il est également important en informatique, notamment dans les théories de la calculabilité et de la complexité. Plus spécifiquement, l'informaticien programmeur se posera souvent la question suivante : toutes les exécutions de tel programme sont-elles finies ?

En dépit des nombreuses justifications de son intérêt, la théorie des ensembles est souvent mal connue des informaticiens. Son enseignement ne figure d'ailleurs pas souvent aux programmes des études en informatique. Cette lacune regrettable est sans doute due au fait que les applications directes de la théorie des ensembles semblent rares. Signalons néanmoins que certains langages de spécification et de programmation (comme Z et SETL) s'inspirent directement de la théorie des ensembles. D'autre part, et c'est beaucoup plus important, certaines démarches de l'informaticien ne trouvent une justification rigoureuse que dans le cadre de cette théorie. On citera en particulier la technique des définitions récursives, qui est à la base de la programmation fonctionnelle. On peut aussi observer que les théories et logiques spéciales utilisées par l'informaticien reposent souvent sur la théorie des ensembles. C'est le cas de la logique de Hoare et de la théorie du point fixe, qui sont abordées dans les chapitres suivants.

La théorie axiomatique des ensembles a fait l'objet de nombreux exposés de qualité. Ces exposés sont habituellement écrits dans un style "hypothético-déductif" : des axiomes sont introduits et des propriétés en sont déduites. Cela signifie que, au fil du temps, l'ensemble des théorèmes s'accroît de façon monotone. Notre but premier n'étant pas la présentation de la théorie des ensembles pour elle-même, notre approche sera différente. On part des concepts les plus simples; ceux-ci donnent lieu à des axiomes "naturels", dont on déduit les premières conséquences. Les problèmes rencontrés nous incitent non seulement à ajouter de nouveaux axiomes, mais aussi à remettre en question des axiomes antérieurement introduits. Ce cheminement est destiné à réconcilier l'approche naïve et l'approche axiomatique, et à montrer que cette dernière est parfaitement compatible avec une intuition éclairée.

Enfin, la logique est pour l'informaticien un outil, et son introduction dans un domaine particulier nécessitera généralement quelques tâtonnements : le choix du langage et des postulats à adopter n'est pas toujours évident. L'exemple de la théorie des ensembles est très instructif à ce point de vue. La notion d'ensemble apparaît dès qu'une collection d'éléments est considérée comme une entité propre. Ce point de vue naïf est développé dans la section 2.2. L'approche constructive, introduite à la section 2.3, augmente la portée de la théorie des ensembles en attribuant le statut d'ensemble aux objets mathématiques usuels (nombres, figures géométriques, fonctions, ...). Elle permet aussi de considérer des ensembles de plus en plus complexes.

L'approche axiomatique formelle complète et précise le point de vue constructif. Dans ce cadre, les objets de type "ensemble" sont eux-mêmes construits au moyen d'une petite collection d'opérateurs bien déterminés, plutôt que d'être "définis" ou donnés a priori. Cette technique indirecte est fréquemment utilisée en informatique, pour définir les types de données (abstraits). Elle est rendue nécessaire par les paradoxes inhérents aux approches précédentes. Le premier apport technique de la théorie des ensembles, et peut-être le plus important, est celui de "nombre infini". Ces nombres généralisés sont étudiés aux dernières sections de ce chapitre; cette étude implique la technique de définition récursive, d'un grand intérêt pour l'informaticien.

2.2. Notion naïve d'ensemble

Un *ensemble* est simplement une collection d'objets, appelés *éléments* de l'ensemble. On introduit le symbole $\in$ pour désigner l'appartenance: l'écriture $a \in E$ signifie que l'objet a est un élément de l'ensemble E.

Il est assez fréquent de réserver les minuscules aux éléments et les majuscules aux ensembles. Cette convention est toutefois d'usage limité. Elle ne permet pas, par exemple, de distinguer les ensembles composés d'éléments de base et les ensembles dont les éléments sont eux-mêmes des ensembles. En géométrie plane, par exemple, on ne confond pas le plan, vu comme un ensemble de points, et l'ensemble des droites du plan.

En général, un domaine d'interprétation de la théorie des ensembles contiendra des ensembles de divers *types*, et aussi des *éléments fondamentaux* qui, tel le point géométrique, ne possèdent pas eux-mêmes d'éléments. Les éléments fondamentaux sont de type 0; les éléments d'un ensemble ou d'une *classe*[2] de type n sont au plus de type $n-1$. Comme il existe une infinité de types, il est pratiquement impossible d'associer une typographie spéciale à chacun d'eux et nous n'essaierons pas. En particulier, une lettre minuscule italique peut désigner un élément ou un ensemble de type quelconque.

On peut considérer que les domaines de la théorie des ensembles se composent d'objets quelconques, puisque tout objet a, semble-t-il, la nature d'un élément ou d'un ensemble. Toute relation binaire sur un domaine peut, a priori, être considérée comme l'interprétation de la relation d'appartenance. On n'exclut donc pas que des formules telles que $x \in x$ ou $x \in y \wedge y \in x$ soient vraies pour certaines interprétations. Formellement, étant donné un domaine U (pour "univers"), un objet x de U est un *élément* s'il existe un objet y de U

[2] Les mots "ensemble" et "classe" sont synonymes dans l'approche naïve mais, dans le cadre axiomatique, la notion de classe est plus générale que celle d'ensemble. Cette distinction ne sera formellement introduite qu'au paragraphe 2.4.3 mais, pour réduire le risque ultérieur de confusion, nous utiliserons dès maintenant le mot "classe" chaque fois que, dans le cadre axiomatique, le mot "ensemble" serait erroné ou gravement restrictif.

tel que $x \in y$;[3] un objet x' de U est un *ensemble* ou une *classe* s'il existe un objet y' de U tel que $y' \in x'$. Un élément fondamental est un élément sans être un ensemble.

Le domaine géométrique euclidien comportera les points du plan (ou de l'espace) comme éléments fondamentaux; les droites, cercles, segments et autres figures seront des ensembles.[4]

2.2.1. *Extension et compréhension*

La propriété permettant de distinguer un ensemble d'un autre genre d'assemblage (tableau, liste, pile, graphe, fonction, etc.) est la suivante: un ensemble est entièrement caractérisé par ses éléments. Cela donne lieu à un critère d'égalité entre ensembles: deux ensembles sont égaux si et seulement s'ils ont les mêmes éléments.[5] Ce critère peut s'écrire

$$\forall x \forall y \, [x = y \ \equiv \ \forall z \, (z \in x \ \equiv \ z \in y)] \, .$$

Cet énoncé est le *principe d'extensionnalité* ; il permet de spécifier un ensemble en énumérant ses éléments (dans un ordre quelconque). Une telle spécification constitue une *définition en extension.* L'écriture

$$E = \{a, b, c\}$$

signifie que E contient exactement les éléments a, b et c. Dans ce contexte, les trois objets a, b et c sont souvent supposés déjà définis, mais ils ne sont pas nécessairement distincts; on souhaite néanmoins que l'hypothèse $a = b$ entraîne $\{a, b, c\} = \{a, c\}$. Remarquons que l'écriture usuelle $\{a, b, c\}$, censée être un terme dénotant l'objet E, n'appartient pas au langage de la logique du premier ordre. On peut cependant admettre qu'il s'agit d'une abréviation, à condition que l'existence et l'unicité du terme dénoté soient garanties; nous reviendrons sur ce point au paragraphe 2.3.4.6. Dans le cas présent, cette condition est formalisée par l'énoncé[6]

$$\exists! y \forall x \, [x \in y \ \equiv \ (x = a \ \vee \ x = b \ \vee \ x = c)] \, .$$

---

[3] Pour des raisons évidentes, on n'emploie pas la formule $x \in U$ pour exprimer la présence de l'objet x dans l'univers U; le symbole d'appartenance note ici une relation binaire sur U.

[4] Signalons déjà que, pour des raisons évoquées plus loin, on se limitera le plus souvent à un univers ne contenant qu'un seul élément fondamental. Le fait de ne contenir aucun élément caractérisera univoquement cet objet *vide*; il n'y aura donc pas d'inconvénient à lui attribuer aussi le statut d'ensemble, ce qui simplifiera la terminologie. Tout objet de l'univers sera alors une classe (mais pas nécessairement un ensemble). Cette restriction, liée à des impératifs techniques, n'empêchera pas la modélisation aisée du domaine géométrique par exemple; les éléments fondamentaux s'avèreront inutiles, tant en pratique qu'en théorie.

[5] L'ensemble est donc l'assemblage le plus simple qui soit; les autres structures sont plus fines, en ce sens que l'identité des composants n'implique pas nécessairement l'identité des structures. En particulier, la liste $(0, 1, 2)$ n'est pas égale à la liste $(0, 2, 1)$.

[6] Rappelons que "$\exists!$" signifie "il existe un et un seul"; $\exists! x \, P(x)$ est une notation abrégée de la formule $\exists x \, [P(x) \wedge \forall y \, (P(y) \Rightarrow y = x)]$.

Cet énoncé est une instance du schéma dit "de compréhension" :

$$\exists! y \forall x [x \in y \equiv P(x)],$$

dans lequel $P(x)$ représente une formule où x n'a que des occurrences libres et où y n'apparaît pas. Une instance valide de ce schéma définit un terme, que l'on peut écrire $\{x : P(x)\}$. (La variable x est liée dans cette notation, comme dans l'instance qui lui a donné naissance.) On a donc

$$\forall x [x \in \{z : P(z)\} \equiv P(x)].$$

La définition en extension n'est pas possible pour un ensemble comportant une infinité d'éléments; même dans le cas fini, elle est souvent très incommode. On préférera donc spécifier un ensemble en donnant une propriété caractéristique de ses éléments, c'est-à-dire une propriété qui est vraie pour tous les éléments de l'ensemble et fausse pour tous les autres objets de l'univers. Une telle spécification constitue une *définition en compréhension.* Ceci nous incite à faire du schéma de compréhension un schéma d'axiome, appelé *principe de compréhension.*[7]

Le principe d'extensionnalité s'applique aux ensembles, mais pas aux éléments fondamentaux. Cette restriction est gênante et même formellement inacceptable si on veut obtenir une théorie du premier ordre. En effet, les éléments fondamentaux sont des objets du domaine d'interprétation, et un axiome de la théorie devrait s'appliquer indistinctement à tous les objets de ce domaine. Si on souhaite garder le principe d'extensionnalité, on ne pourra admettre qu'un seul élément fondamental dans un domaine, et ce sera nécessairement l'ensemble vide. Cela ne nuira pas à l'universalité de la théorie des ensembles; en particulier, nous verrons plus loin comment on peut modéliser les points de la géométrie et d'autres objets mathématiques classiques par des ensembles.[8]

2.2.2. *Algèbre ensembliste*

A partir des principes d'extensionnalité et de compréhension, on peut introduire facilement les constantes prédicatives et fonctionnelles habituelles dans le cadre des ensembles. Dans le tableau suivant, nous ne reprenons que les constantes représentant les notions les plus importantes; les expressions de gauche sont considérées comme des abréviations des expressions de droite correspondantes.

$$\begin{array}{lll} \{a, b, \ldots, n\} & \text{pour} & \{x : (x = a \vee x = b \vee \cdots \vee x = n)\}, \\ a \subset b & \text{pour} & \forall x [x \in a \Rightarrow x \in b], \end{array}$$

[7] Signalons déjà que ce principe sera remis en question plus loin, et remplacé par un principe plus faible.

[8] Voilà une première raison pour éliminer la notion d'élément fondamental.

$$
\begin{array}{lll}
a \cup b & \text{pour} & \{x : (x \in a \vee x \in b)\}, \\
a \cap b & \text{pour} & \{x : (x \in a \wedge x \in b)\}, \\
a \setminus b & \text{pour} & \{x : (x \in a \wedge x \notin b)\}, \\
\emptyset & \text{pour} & \{x : \mathit{false}\}, \\
V & \text{pour} & \{x : \mathit{true}\}, \\
\mathcal{P}(a) & \text{pour} & \{x : x \subset a\}.
\end{array}
$$

Remarques. Dans ce volume, l'implication sera dénotée par le symbole $\Rightarrow$ au lieu du symbole $\supset$, pour éviter toute confusion avec l'inclusion et la contenance ensemblistes. On écrit parfois $b \ni a$ pour $a \in b$ et $b \supset a$ pour $a \subset b$.[9] De même, on écrit $a \notin b$ et $a \not\subset b$ pour $\neg(a \in b)$ et $\neg(a \subset b)$, respectivement. La notation $\mathcal{P}(a)$ désigne l'ensemble des *parties* ou *sous-ensembles* de a. On a par exemple $\mathcal{P}(\{b,c\}) = \{\emptyset, \{b\}, \{c\}, \{b,c\}\}$.

L'associativité et la commutativité des opérateurs $\cup$ (réunion) et $\cap$ (intersection) découlent immédiatement des propriétés analogues des connecteurs $\wedge$ et $\vee$. Ces propriétés permettent de donner un sens à une expression telle que $a_1 \cup a_2 \cup \cdots \cup a_n$, que nous écrirons plutôt $\bigcup\{a_1, a_2, \ldots, a_n\}$. La *réunion généralisée* et l'*intersection généralisée* peuvent aussi être définies directement; on a

$$
\begin{array}{ll}
\bigcup a & =_{def} \{x : \exists y\,(y \in a \wedge x \in y)\}, \\
\bigcap a & =_{def} \{x : \forall y\,(y \in a \Rightarrow x \in y)\}.
\end{array}
$$

Ces définitions[10] n'exigent pas que l'argument a soit de type $\{a_1, \ldots, a_n\}$; a est n'importe quel ensemble dont les éléments sont eux-mêmes des ensembles.[11] Le cas où a est l'ensemble vide est intéressant; on a

$$
\begin{array}{l}
\bigcup \emptyset = \{x : \exists y\,(y \in \emptyset \wedge x \in y)\} = \{x : \exists y\,(\mathit{false})\} = \{x : \mathit{false}\} = \emptyset, \\
\bigcap \emptyset = \{x : \forall y\,(y \in \emptyset \Rightarrow x \in y)\} = \{x : \forall y\,(\mathit{true})\} = \{x : \mathit{true}\} = V.
\end{array}
$$

L'ensemble V contient donc tous les ensembles.

A partir de ces notions, on peut développer l'algèbre ensembliste; à titre d'exemple, considérons l'énoncé

$$\forall x \forall y\,[(x \cap y) = (y \cap x)],$$

qui exprime la commutativité de l'opération d'intersection. Cet énoncé est évidemment un théorème; en effet, on peut successivement le récrire comme suit :

$$
\begin{array}{l}
\forall x \forall y\,[(x \cap y) = (y \cap x)], \\
\forall x \forall y \forall u\,[u \in (x \cap y) \equiv u \in (y \cap x)], \\
\forall x \forall y \forall u\,[u \in \{v : (v \in x \wedge v \in y)\} \equiv u \in \{v : (v \in y \wedge v \in x)\}], \\
\forall x \forall y \forall u\,[(u \in x \wedge u \in y) \equiv (u \in y \wedge u \in x)].
\end{array}
$$

[9] Certains auteurs écrivent $a \subseteq b$ au lieu de $a \subset b$.

[10] en fait, des abréviations.

[11] Voilà une deuxième raison pour éliminer la notion d'élément fondamental; l'argument a sera alors n'importe quel objet de l'univers.

2.2.3. *Algèbre relationnelle*

La notion d'ensemble n'est vraiment utile que si on dispose aussi de la notion de *relation.* (On se limite ici aux relations binaires.) De même qu'un ensemble est caractérisé par ses éléments, une relation est caractérisée par ses *couples.* On obtient un couple (a, b) à partir de deux éléments a et b; a est l'*origine* du couple, b en est l'*extrémité.* On introduit l'égalité entre couples par l'axiome

$$\forall x \forall y \forall u \forall v \, [(x, y) = (u, v) \equiv (x = u \wedge y = v)] \, .$$

On introduit aussi les prédicats *couple*, *rel* et *fonc* par les axiomes

$$\forall x \, [couple(x) \equiv \exists y \exists z \, (x = (y, z))] \, ,$$
$$\forall x \, [rel(x) \equiv \forall y \, (y \in x \Rightarrow couple(y))] \, ,$$
$$\forall x \, [fonc(x) \equiv (rel(x) \wedge \forall u \forall y \forall z \, [((u, y) \in x \wedge (u, z) \in x) \Rightarrow y = z])] \, .$$

Une relation est un ensemble de couples; on écrira donc $(a, b) \in R$, par exemple, mais l'écriture aRb (parfois ambiguë) est aussi d'usage fréquent.[12] Classiquement, le *domaine* d'une relation est l'ensemble des origines des couples et le *codomaine* est l'ensemble des extrémités. Cela peut se formaliser :

$$dom(a) =_{def} \{x : rel(a) \wedge \exists y[(x, y) \in a]\} \, ,$$
$$cod(a) =_{def} \{x : rel(a) \wedge \exists y[(y, x) \in a]\} \, .$$

Enfin, la relation f est une *fonction*, ce que l'on note parfois $fonc(f)$, si pour tout élément x du domaine de f, il existe un seul élément y du codomaine tel que $(x, y) \in f$. Cet élément y est noté $f(x)$; c'est *l'image* de x par f. On note $f|_e$ la *restriction* de f à e; on a $(x, y) \in f|_e$ si $(x, y) \in f$ et $x \in e$. (Cette notion s'étend aux relations. Elle est généralement utilisée quand e est un sous-ensemble du domaine de f.) L'ensemble des fonctions de domaine a et de codomaine inclus dans b est souvent noté $a \to b$.

---

[12] Assimiler une relation à un ensemble de couples peut paraître assez réducteur; s'il est clair qu'il n'y a pas de relation sans un ensemble de couples, il est moins clair qu'une relation, au sens habituel du terme, ne soit rien de plus qu'un ensemble de couples. Considérons par exemple les relations "est conjoint de" et "danse avec". Il est très possible qu'en un lieu donné (une piste de danse) et à un instant donné, ces deux relations soient égales au sens de la théorie des ensembles. Dans le langage courant, néanmoins, elles ne le sont pas, tout simplement parce qu'elles sont *définies* en termes très différents. Le logicien rendra compte de cette nuance importante en qualifiant les deux relations de *coextensives*; les *extensions* des deux relations sont égales, leurs *intensions* diffèrent. On verra dans le volume III de cet ouvrage que la notion d'intension (qui marque l'*intention* de l'agent définissant) est parfaitement mathématisable. En mathématique existe néanmoins une très forte tendance à négliger l'intension et à assimiler les objets à leur extension. Cette tendance est relativement récente, et a crû parallèlement au développement de la théorie des ensembles; bien plus, elle est à la base de cette théorie, et une théorie non extensionnelle des fonctions (telle la théorie des catégories) est inévitablement plus compliquée que la théorie extensionnelle, maintenant classique, dont nous abordons le développement ici. Cela n'a rien d'étonnant : la théorie extensionnelle se base sur un concept très simple — grossièrement simplifié ? — de fonction. Cette simplification s'est avérée extrêmement féconde. On peut en rapprocher la notion d'implication matérielle, suffisante en mathématique, mais simplification extrême de la notion usuelle d'implication.

Soit f une fonction de domaine a et de codomaine b; f est une *bijection* si pour tout élément $y \in b$ il existe exactement un élément $x \in a$ tel que $f(x) = y$. La fonction qui à tout $y \in b$ associe cet élément $x \in a$ se note f^{-1}; c'est une bijection de domaine b et de codomaine a. Le concept de bijection est important, parce qu'il permet de faire abstraction des objets manipulés pour se concentrer sur les lois permettant de les combiner.[13] Une bijection de domaine a et de codomaine b permet, dans une certaine mesure que nous préciserons plus loin, d'identifier les ensembles a et b. Bien plus, on peut voir les éléments de ces ensembles non comme des objets, mais comme des noms d'objets; selon ce point de vue, $x \in a$ et $f(x) \in b$ sont des noms (dans deux "langues" différentes) d'un même objet, non autrement précisé. Une bijection apparaît alors comme une fonction de traduction, ou de "renommage".

On vient de définir la fonction comme un cas particulier de la relation; dans une fonction, l'origine d'un couple détermine son extrémité. On peut aussi adopter le point de vue inverse et dériver la notion de relation de celle de fonction. Une relation de domaine a et de codomaine b est un cas particulier de fonction de domaine a et de codomaine inclus dans $\mathcal{P}(b)$. Si r est une relation de domaine a et de codomaine b, la fonction correspondante f est déterminée par l'égalité $f(x) = \{y : (x,y) \in r\}$; en pratique, on ne changera pas de notation et on écrira $r(x)$ au lieu de $f(x)$. Plus généralement, si a' est un sous-ensemble de a, on écrira $r(a') = \{y : \exists x\,[x \in a' \wedge (x,y) \in r]\}$. La relation r détermine donc une deuxième fonction, de domaine $\mathcal{P}(a)$ et de codomaine inclus dans $\mathcal{P}(b)$. Le plus souvent, il n'y a pas d'inconvénient à utiliser un même symbole pour désigner une relation r et les deux fonctions qui en dérivent naturellement.

La *relation réciproque* de r est notée r^{-1}. Son domaine est b et son codomaine est a; on a

$$\forall x \forall y\,[(x,y) \in r^{-1} \equiv (y,x) \in r]\,.$$

Le symbole r^{-1} désignera non seulement cette relation, mais aussi les deux fonctions dérivées, de domaines respectifs b et $\mathcal{P}(b)$, et de codomaine inclus dans $\mathcal{P}(a)$. Dans le cas particulier d'une fonction, la notation f^{-1} désignera

[13] Considérons un exemple bien connu en analyse. La fonction e qui à tout nombre réel x associe son exponentielle $e(x) = e^x$ est une bijection de domaine $\mathbf{R}$ et de codomaine $\mathbf{R}_0^+$ (réels strictement positifs). La propriété bien connue $e^{x+y} = e^x.e^y$ montre que l'addition des réels et la multiplication des réels positifs, quel que soit l'intérêt pratique de ces opérations, ne sont que deux instances d'un même concept, plus général donc, mathématiquement parlant, plus intéressant, celui de structure de groupe abélien. La fonction exponentielle est un *isomorphisme* du groupe abélien ordonné $(\mathbf{R}, 0, +, \leq)$ sur le groupe abélien ordonné $(\mathbf{R}_0^+, 1, ., \leq)$. La notion d'isomorphisme est *générique* : elle dépend de la structure algébrique sous-jacente. Par exemple, la fonction successeur, qui à tout entier x associe l'entier $x+1$, est un isomorphisme de l'ensemble ordonné $(\mathbf{Z}, \leq)$ sur lui-même, mais n'est pas un isomorphisme du groupe $(\mathbf{Z}, +)$ sur lui-même. En effet, le successeur d'une somme n'est pas égal à la somme des successeurs.

en outre une fonction de domaine inclus dans b et de codomaine a. Cette troisième acception supplémentaire (que nous avons déjà introduite dans le cas particulier de la bijection) revient à admettre l'écriture $f^{-1}(y) = x$, en plus des écritures $f^{-1}(y) = \{x\}$ et $f^{-1}(\{y\}) = \{x\}$, pour tout couple $(x, y) \in f$.

La *composition* des relations binaires r et s est une relation binaire $t = r \circ s$ telle que $(x, z) \in t$ si et seulement s'il existe y tel que $(x, y) \in s$ et $(y, z) \in r$ (attention à l'ordre des opérandes; la composition relationnelle n'est pas commutative). Si $r \circ s$ n'est pas vide, alors $cod(s) \cap dom(r)$ n'est pas vide. Si r, s sont des fonctions, si $x \in dom(s)$ et $s(x) \in dom(r)$, alors $(r \circ s)(x) = r(s(x))$.

Dans la plupart des domaines mathématiques, le concept de fonction semble essentiel et celui de relation, accessoire. Nous venons de voir que la théorie des ensembles ne cautionne pas réellement cette tendance. D'une part, chaque concept peut être présenté à partir de l'autre et, d'autre part, le mathématicien confronté à une fonction trouvera souvent profitable d'envisager la généralisation à une relation. Ces deux points sont aussi d'application en informatique. Les fonctions modélisent les programmes déterministes et les relations modélisent les programmes non déterministes.

A côté d'avantages importants, la conception extensionnelle des relations et des fonctions présente aussi quelques inconvénients pratiques. Voici d'abord un exemple mathématique. En analyse classique, on parle souvent de "fonctions de $\mathbf{R}$ dans $\mathbf{R}$ (ou de $\mathbf{C}$ dans $\mathbf{C}$)" voire de "fonctions réelles (ou complexes)" pour désigner des fonctions dont le domaine et le codomaine sont des *sous-ensembles* de $\mathbf{R}$ (ou de $\mathbf{C}$). Le point de vue extensionnel ne permet pas de distinguer immédiatement la fonction Γ, dont le domaine "exact" est $\mathbf{C} \setminus (\mathbf{Z} \setminus \mathbf{N}_0)$, d'une hypothétique "fonction incomplète" dont le "domaine" serait $\mathbf{C}$. Le point de vue intensionnel, classique en analyse jusqu'à une époque relativement récente, n'a pas ce défaut; on peut écrire "une fonction de $\mathbf{C}$ dans $\mathbf{C}$ est une *loi* qui, à tout nombre complexe, associe *0 ou 1* nombre complexe".

Il n'est nullement indispensable de sortir du cadre de la théorie des ensembles pour prendre en compte des éléments intensionnels du concept de fonction, dans les cas relativement rares où cela est nécessaire. En particulier, on peut associer à une fonction un "ensemble de référence" contenant son domaine.[14]

[14] Plus précisément, une *correspondance* est un triplet ordonné $C = (a, r, b)$ où r est une relation dont le domaine est inclus dans a et dont le codomaine est inclus dans b. Si r est une fonction, la correspondance C est *fonctionnelle*. La théorie des fonctions complexes d'une variable complexe est donc la théorie des correspondances fonctionnelles de type $(\mathbf{C}, f, \mathbf{C})$. Le plus souvent, le concept de correspondance n'est pas utilisé explicitement; on dira plutôt, par exemple, que Γ est une *fonction partielle* sur $\mathbf{C}$ tandis que l'identité, les polynômes et l'exponentielle sont des *fonctions totales* sur $\mathbf{C}$. Dans ce contexte, on emploie souvent "fonction" pour "fonction partielle". Une correspondance (a, f, b) est une *application* si f est une fonction de domaine a, c'est-à-dire si pour tout $x \in a$ il existe un et un seul $y \in b$ tel que $(x, y) \in f$. L'application (a, f, b) est *injective* (*surjective*) si pour tout $y \in b$ il existe au plus (au moins) un $x \in a$ tel que $(x, y) \in f$. L'application (a, f, b) est *bijective* si elle est à la fois injective et surjective; c'est le cas si f est une bijection de domaine a et de codomaine b.

Une autre technique classique permet de transformer une fonction f de domaine inclus dans a en une fonction (toujours notée f) dont le domaine est exactement a. Pour ce faire, on introduit un objet fictif, noté $\perp$ (lire "bottom" ou "non défini"), et on pose $f(x) = \perp$ pour tout $x \in a \setminus dom(f)$. On obtient ainsi une fonction (totale) de domaine a et de codomaine inclus dans $b \cup \{\perp\}$.[15] Cette technique est fréquente en informatique mais plus rare en mathématique; on ne l'utilise pas en analyse car l'adjonction de l'élément $\perp$ ferait perdre aux ensembles et fonctions concernés des propriétés utiles. Nous reviendrons sur tout cela au chapitre 5.

On peut introduire simplement les propriétés classiques des relations: (anti-)réflexivité, (anti-)symétrie, et transitivité, et donner ainsi naissance à l'algèbre relationnelle.

$$\begin{aligned}
&\forall x\,[refl(x) \equiv (rel(x) \wedge \forall y \forall z\,[(y,z) \in x \Rightarrow \{(y,y),(z,z)\} \subset x])]\,,\\
&\forall x\,[antirefl(x) \equiv (rel(x) \wedge \forall y\,[(y,y) \notin x])]\,,\\
&\forall x\,[sym(x) \equiv (rel(x) \wedge \forall y \forall z\,[(y,z) \in x \Rightarrow (z,y) \in x])]\,,\\
&\forall x\,[antisym(x) \equiv (rel(x) \wedge \forall y \forall z\,[\{(y,z),(z,y)\} \subset x \Rightarrow y = z])]\,,\\
&\forall x\,[trans(x) \equiv (rel(x) \wedge \forall y \forall z \forall w\,[\{(y,z),(z,w)\} \subset x \Rightarrow (y,w) \in x])]\,,\\
&\forall x\,[equiv(x) \equiv (refl(x) \wedge sym(x) \wedge trans(x))]\,,\\
&\forall x\,[ordre(x) \equiv (refl(x) \wedge antisym(x) \wedge trans(x))]\,.
\end{aligned}$$

On appelle *préordre* une relation réflexive et transitive; une *équivalence* est un préordre symétrique et un *ordre* est un préordre antisymétrique. Il n'est pas nécessaire d'exiger la réflexivité d'une équivalence; on démontre facilement la formule

$$\forall x\,[(sym(x) \wedge trans(x)) \Rightarrow refl(x)]\,.$$

Une relation d'équivalence r sur a (telle que $dom(r) = cod(r) = a$) détermine une *partition* de a, c'est-à-dire une répartition des éléments de a en sous-ensembles exhaustifs et mutuellement exclusifs. La relation d'équivalence "a la même nationalité que" partitionne la population humaine en nations.

Une relation d'ordre s sur a correspond bien à la notion intuitive d'ordre; observons néanmoins qu'un ordre n'est pas toujours *total*: il peut exister $b, c \in a$ tels qu'aucun des couples (b,c) et (c,b) n'appartienne à s. Dans la population humaine, la relation "est un ancêtre de" est antisymétrique et transitive, mais non totale; dans $\mathcal{P}(a)$, la relation d'inclusion est également un ordre partiel dès que a comporte au moins deux éléments. La relation "plus petit ou égal" est un ordre total sur l'ensemble des nombres entiers. On ne rappelle pas ici le vocabulaire classique associé aux relations d'ordre (minorant, minimum, etc.); signalons seulement un usage commode et fréquent: on écrit, par exemple, "s-minorant" si on souhaite souligner que la relation d'ordre sous-jacente est s.

[15] Il est parfois commode d'adjoindre $\perp$ au domaine, de poser $f(\perp) = \perp$ et d'obtenir ainsi une fonction de domaine $a \cup \{\perp\}$ et de codomaine inclus dans $b \cup \{\perp\}$.

Un ensemble est une relation si tous ses éléments sont des couples; c'est visiblement le cas de l'ensemble vide. De la formule $(y,z) \notin \emptyset$, on déduit immédiatement $(y,z) \in \emptyset \Rightarrow \{(y,y),(z,z)\} \subset \emptyset$; l'ensemble vide est donc une relation réflexive. On vérifie immédiatement que c'est aussi une relation antiréflexive, symétrique, antisymétrique et transitive, d'où aussi une relation d'ordre et une relation d'équivalence; enfin, c'est aussi une fonction.

Toutes ces notions sont couramment utilisées pour décrire des relations sur des ensembles donnés. Considérons par exemple l'ensemble des entiers. Sur cet ensemble, la relation "est successeur de" est fonctionnelle, antiréflexive, antisymétrique et non transitive. La relation "est plus petit que" est non fonctionnelle, antiréflexive, antisymétrique et transitive. La relation "a même parité que" est non fonctionnelle, réflexive, symétrique et transitive. La relation "divise" est non fonctionnelle, réflexive et transitive (préordre); elle n'est pas symétrique mais elle n'est pas non plus antisymétrique: deux entiers opposés se divisent mutuellement.

2.2.4. *Produit d'ensembles*

La notion usuelle de produit cartésien donne lieu à la définition

$$\times(a,b) =_{def} \{x : \exists u \exists v [x = (u,v) \wedge u \in a \wedge v \in b]\}.$$

Conformément à l'usage, on écrit $a \times b$ plutôt que $\times(a,b)$. Un élément du produit cartésien de a et b est donc un couple dont l'origine appartient à a et l'extrémité appartient à b. Cette notion donne lieu à de nombreux résultats. A titre d'exemple, on a les théorème suivant:

$$\forall x \forall y \forall z [(rel(x) \wedge dom(x) \subset y \wedge cod(x) \subset z) \equiv x \subset y \times z].$$

Cet énoncé exprime qu'une relation est un sous-ensemble du produit de son domaine et de son codomaine.

Il est intéressant de généraliser la notion de produit, comme on a généralisé celles de réunion et d'intersection. Une première technique consiste à définir d'abord la notion de n-uplet ordonné. Le triplet ordonné (a_1,a_2,a_3) est défini comme le couple $((a_1,a_2),a_3)$ et, plus généralement, pour tout $n > 2$ on pose

$$(a_1,\ldots,a_n) =_{def} ((a_1,\ldots,a_{n-1}),a_n).$$

(Le problème de la validité des définitions de ce type, dites *récursives*, sera abordé au paragraphe 2.3.4.) Si par exemple l'ensemble E a pour éléments les ensembles A_1, A_2, A_3, A_4 et A_5, on notera $\mathsf{X}(E)$ l'ensemble des quintuplets ordonnés (a_1,a_2,a_3,a_4,a_5), où $a_i \in A_i$, pour tout $i = 1,\ldots,5$. Le même ensemble sera noté aussi $\mathsf{X}_{i\in I} A_i$, si I est l'ensemble $\{1,2,3,4,5\}$.

L'ensemble $\mathsf{X}(E)$ est défini pour tout ensemble E fini, dont les éléments sont des ensembles. Il est vide si et seulement si l'un des éléments de E est vide.

L'ensemble $\mathsf{X}(\emptyset)$ comporte un seul élément, le 0-uplet $()$.[16]

Cette technique ne s'applique qu'aux ensembles finis et n'est donc pas suffisante. On obtient une notion plus générale en observant qu'un élément du *produit* (à définir) de E devrait "se composer" d'un élément quelconque de chacun des éléments de E. Il faut naturellement que l'élément particulier de E dont provient un composant du produit soit identifiable. L'ensemble $\prod E$ contiendra donc les *fonctions* qui, à chaque élément x de E, associent un élément de x. De telles fonctions s'appellent *fonctions de choix* sur E. On reviendra sur cette notion au paragraphe consacré à l'axiome du choix. Avec cette formulation, l'ensemble $\prod\{a,b\}$ n'est pas égal à l'ensemble $a \times b$, mais il existe une bijection naturelle entre eux. Si la fonction f est un élément de $\prod\{a,b\}$, alors $(f(a), f(b)) \in a \times b$. Réciproquement, si $(x,y) \in a \times b$, alors $\{(a,x),(b,y)\} \in \prod\{a,b\}$.

On peut assimiler $\prod\{a,b\}$ à $a \times b$, mais on ne peut pas assimiler $\prod\{a,a\}$ à $a \times a$, puisque $\{a,a\} = \{a\}$. Ce problème n'est qu'apparent : on le résout en "indexant" différemment les deux copies de a. Formellement, cela peut se faire en utilisant le produit cartésien binaire : une bijection naturelle existe entre $\prod\{\{0\} \times a, \{1\} \times a\}$ et $a \times a$. Plus généralement, si I est un ensemble et si, pour tout $i \in I$, A_i est un ensemble, on notera $\prod_{i\in I} A_i$ l'ensemble dont les éléments sont les fonctions de domaine I qui à tout $i \in I$ associent un élément $a_i \in A_i$. Même si les A_i sont deux à deux distincts, il n'y a pas égalité entre $\prod_{i\in I} A_i$ et $\prod\{A_i : i \in I\}$ mais cela n'est pas gênant : l'important est la bijection naturelle simple qui existe entre les deux ensembles.[17]

En pratique, les distinctions entre les variantes du produit cartésien sont négligées. Si $I = \{1,2,3,4,5\}$, la notation (a_1,a_2,a_3,a_4,a_5) désignera indifféremment un élément de $\mathsf{X}_{i\in I} A_i$, ou de $\prod_{i\in I} A_i$, ou de $\prod\{A_i : i \in I\}$. Dans le même ordre d'idée, l'écriture $A_1 \times \cdots \times A_5$ est fréquente même si, formellement, l'opérateur de produit cartésien binaire n'est pas associatif.[18]

---

[16] La nature exacte de l'unique 0-uplet n'a pas été définie; le seul point important est que $\mathsf{X}(\emptyset)$ contienne un seul élément. De même, si A est un ensemble, $\mathsf{X}(\{A\})$ est l'ensemble caractérisé par la condition $(a) \in \mathsf{X}(\{A\})$ si et seulement si $a \in A$. La nature exacte de (a) n'est pas définie; le point important est que A et $\mathsf{X}(\{A\})$ sont en bijection. Par souci d'exhaustivité, on fixera $() = \emptyset$ et $(a) = a$ pour tout a, d'où $\mathsf{X}(\emptyset) = \{\emptyset\}$ et $\mathsf{X}(\{A\}) = A$.

[17] Nous voyons un exemple du caractère "abstractif" des bijections : les ensembles $\prod_{i\in I} A_i$ et $\prod\{A_i : i \in I\}$ comportent les noms, dans deux langages (un peu) différents, des éléments d'une entité abstraite, non autrement décrite, qui est le produit cartésien des ensembles A_i. Ce produit n'est pas tellement intéressant par nature, mais plutôt par ses interactions avec ses composantes. Plus précisément, le point essentiel est que, au départ d'un élément x du produit et d'une composante A_i, on détermine un et un seul élément $y \in A_i$; que cet élément soit noté $x(i)$ ou $x(A_i)$ est accessoire.

[18] Remarquons que la notion de produit cartésien généralisé (utilisant le symbole Π) n'exige pas le recours, illicite à ce stade, à la récursivité; cette notion doit donc supplanter celle de produit en croix, symbolisée par X.

Le cas où I est un ensemble totalement ordonné et où $A_i = A$ pour tout $i \in I$ est spécialement important. Les éléments de l'ensemble $\prod_{i \in I} A$ sont les *suites* ou *séquences*[19] sur A indexées par I. On parle de suites finies ou infinies selon que I est fini ou infini. Une suite finie sur A indexée par un ensemble de n éléments est assimilée à un n-uplet ordonné et notée comme tel. Si l'ensemble d'index I est l'ensemble $\mathbf{N}$ des naturels, on parle simplement de suites (infinies) sur A. On utilise parfois l'écriture $\{(0, a_0), \ldots, (n, a_n), \ldots\}$ pour désigner une suite sur A, mais plus souvent l'écriture $(a_0, \ldots, a_n, \ldots)$ ou encore $(a_n : n \in \mathbf{N})$. L'abréviation (a_n) est fréquente quand elle n'introduit pas d'ambiguïté.

2.2.5. *Relation d'équivalence et ensemble quotient*

Le mécanisme d'abréviation nous permet de définir de nouvelles méthodes pour obtenir des ensembles à partir d'autres ensembles. On l'utilise pour introduire l'une des constructions les plus utiles de l'algèbre relationnelle, le *quotient* d'un ensemble par une relation d'équivalence.

Cette construction peut paraître, de prime abord, assez compliquée et abstraite. Elle est pourtant motivée par des problèmes bien réels. En premier lieu, il convient d'observer que la notion d'équivalence généralise celle d'égalité. On dit informellement que deux objets sont équivalents s'ils ont une certaine propriété en commun. Etant donné un ensemble relativement grand d'objets, il est commode de classer ces objets en catégories. Si on considère l'ensemble des êtres humains, on peut envisager une classification d'après la nationalité (en supposant que chaque être humain ait une et une seule nationalité). La relation "a la même nationalité que" est visiblement réflexive, symétrique et transitive : c'est une relation d'équivalence. Les différentes nationalités elles-mêmes forment un ensemble, qui est le quotient de l'ensemble des êtres humains par la relation d'équivalence "a la même nationalité que". Chaque nation forme une *classe d'équivalence*.[20]

Un exemple plus mathématique, d'où dérive le mot "quotient", est celui de la division arithmétique exacte. Soit a l'ensemble des six premiers nombres naturels et b la relation "a la même parité que". Le quotient a/b sera $\{\{0,2,4\},\{1,3,5\}\}$. Il compte deux éléments comportant chacun trois éléments, ce qui illustre le fait que le quotient de 6 par 2 est 3. Si on remplace l'ensemble a par l'ensemble de tous les entiers, le quotient comportera encore deux éléments, l'ensemble des nombres pairs et l'ensemble des nombres impairs.

Passons maintenant à l'aspect formel. On définit successivement

$$mod(a,b) =_{def} \{x : equiv(b) \wedge a \in dom(b) \wedge (a,x) \in b\},$$
$$c/b =_{def} \{x : equiv(b) \wedge dom(b) = c \wedge \exists y\,[y \in c \wedge x = mod(y,b)]\}.$$

[19] Dans les développements formels, on réservera le mot "suite" au cas (très fréquent) où I est bien ordonné (§ 2.2.7).

[20] Le concept de classe d'équivalence est indépendant de la distinction entre ensemble et classe qui sera introduite plus loin.

L'ensemble $mod(a, b)$ est la *classe d'équivalence* de a par rapport à b, c'est-à-dire l'ensemble des éléments du domaine de b qui sont en relation avec a; si a n'appartient pas au domaine de b, l'ensemble $mod(a, b)$ est vide.
Le quotient c/b d'un ensemble c par une relation d'équivalence b de domaine c est l'ensemble des classes d'équivalence des éléments de c par rapport à b.
Cette opération ensembliste est très fréquemment utilisée pour introduire de nouveaux objets. Quelques exemples seront présentés dans les paragraphes suivants.

2.2.6. *Eléments*

L'algèbre ensembliste et l'algèbre relationnelle, pour intéressantes qu'elles soient, ne permettent pas l'étude d'ensembles et de relations particuliers. En effet, à ce stade, le langage de la théorie ne permet pas de noter les éléments des ensembles usuels, par exemple les nombres entiers ou les points de l'espace euclidien.

La solution naturelle à ce problème consiste à enrichir le langage au fur et à mesure que le besoin s'en fait sentir. On peut par exemple ajouter à la signature[21] du langage la constante individuelle 0 et la constante fonctionnelle monadique *succ* (pour successeur); cela permet de noter les nombres naturels. On pourrait aussi ajouter immédiatement toute la théorie des nombres entiers; cela permettrait d'étudier, dans une certaine mesure, les ensembles de nombres, comme l'ensemble des nombres premiers.

Cette approche conduit rapidement à une théorie pléthorique. Dans la mesure où les théories formelles ont pour but de clarifier, de préciser et d'unifier les théories mathématiques usuelles, une telle approche est tout à fait inappropriée. Une solution plus satisfaisante sera présentée à la section 2.3.

2.2.7. *Théorie des ensembles et sémantique*

La sémantique du calcul des prédicats et des théories du premier ordre est basée sur les notions de structure et d'interprétation. Ces notions font appel aux rudiments de la théorie des ensembles. Considérons à nouveau le cas de la théorie de l'ordre (y compris ses extensions), introduite au chapitre I.9. Une structure (E, R) est un modèle de la théorie de l'ordre si R est une relation d'ordre sur E. Si on admet une intervention plus importante de la théorie des ensembles, la notion d'ordre peut être étudiée plus en profondeur.

---

[21] Rappelons que la *signature* d'un langage, ou d'une théorie dont les axiomes appartiennent à ce langage, est l'ensemble des symboles *extralogiques* du langage. Ceux-ci sont les constantes individuelles, fonctionnelles et prédicatives du langage, par opposition aux symboles logiques, qui sont les variables (individuelles), les connecteurs propositionnels et les quantificateurs. Les parenthèses, crochets, virgules et autres séparateurs sont des symboles auxiliaires dont l'usage peut être évité par l'emploi des notations préfixée ou postfixée.

Dans le cadre de la logique du premier ordre, seuls les objets du domaine d'interprétation peuvent être évoqués, à l'exclusion des collections d'objets. Dans le cadre de la théorie des ensembles, cette restriction subsiste, mais il est possible d'évoquer simultanément un ensemble E et un élément a de cet ensemble, en considérant que E et a sont deux objets de l'univers.

Cette possibilité spécifique de la théorie des ensembles est remarquablement utile. La modélisation de la notion de bon ordre, en particulier, devient évidente. Rappelons que (E, R) est un *bon ordre*[22] si toute partie non vide $A \subset E$ admet un minimum, c'est-à-dire un élément m tel que l'on ait mRa pour tout $a \in A$. La formule caractérisant le bon ordre est donc

$$\forall A\,(A \in \mathcal{P}(E) \setminus \{\emptyset\} \;\Rightarrow\; \exists m\,[m \in A \wedge \forall a\,(a \in A \;\Rightarrow\; (m,a) \in R)])\,.$$

Un ordre (E, R) est *bien fondé* si toute partie non vide A de E admet un élément minimal, c'est-à-dire un élément m' tel que l'on ait $\neg aRm'$ pour tout $a \in A \setminus \{m'\}$. Un bon ordre est donc simplement un ordre bien fondé et total. On a aussi le théorème suivant, dont nous verrons plus loin qu'il permet de justifier les définitions récursives, en mathématique et en informatique.

Théorème 2.1. Un ensemble ordonné (E, R) est bien fondé si et seulement si aucune suite infinie strictement R-décroissante ne peut être formée avec les éléments de E.

Preuve. Soit $(a_n : n \in \mathbf{N})$ une suite d'éléments distincts de l'ensemble bien fondé E. L'ensemble $\{a_n : n \in \mathbf{N}\}$ admet un élément minimal a_k, ce qui interdit $a_{k+1}Ra_k$; la suite (a_n) n'est pas R-décroissante. Réciproquement, soit A un sous-ensemble non vide, dépourvu d'élément minimal, de l'ensemble non bien fondé E. Soit $a_0 \in A$; puisque a_0 n'est pas minimal, il existe $a_1 \in A \setminus \{a_0\}$ tel que a_1Ra_0; de même, puisque a_1 n'est pas minimal, il existe $a_2 \in A \setminus \{a_0, a_1\}$ tel que a_2Ra_1. On construit ainsi une suite infinie strictement R-décroissante dans A et dans E.

Remarque. Le procédé de construction de la suite (a_n) est clair mais ne constitue pas une définition *formelle* de cette suite tant que la notion de définition récursive n'a pas été incluse dans la théorie; en outre, a_n n'est pas univoquement défini en fonction de A et de $\{a_0, \ldots, a_{n-1}\}$, ce qui indique une utilisation implicite de l'axiome du choix. On n'utilisera pas le théorème avant d'avoir réglé ces deux questions.

Les notions de minorant et de meilleure borne inférieure se formalisent tout aussi aisément. On peut aussi étudier pour eux-mêmes les sous-ensembles d'un ensemble ordonné. Par exemple, un sous-ensemble A de E est une *chaîne* s'il est totalement ordonné pour R. On peut également évoquer et étudier les fonctions entre ensembles ordonnés. Le chapitre 5, consacré à la théorie du point fixe, fera usage de cette possibilité.

[22] on dit aussi que E est *bien ordonné* par R.

2.3. Approche constructive

Même sous sa forme naïve, la théorie des ensembles est utile parce que la notion d'ensemble généralise, abstrait et unifie les diverses techniques de combinaison d'objets utilisées en mathématique et en informatique. Le champ d'action de la théorie sera notablement élargi si les objets eux-mêmes, et non seulement les combinaisons de ceux-ci, peuvent être modélisés par des ensembles. La représentation d'objets par d'autres objets est une technique courante en informatique; nous passons brièvement en revue les moyens utilisés habituellement, avant de les appliquer à la modélisation ensembliste des objets mathématiques tels les nombres et les figures géométriques.

2.3.1. *L'approche constructive concrète en informatique*

Les "objets informatiques" sont les types de données (nombres, mots, tables, listes, piles, graphes, etc.) que le programmeur utilise pour poser des problèmes et construire des algorithmes de résolution pour ces problèmes. Lors de la traduction de l'algorithme en un programme exécutable, il est nécessaire de convertir tous ces types de données en les quelques types admis par le langage de programmation utilisé. Dans les langages évolués, cette tâche est largement automatisée. Dans les langages plus anciens, comme le Fortran, ce travail est fait par le programmeur.

Dans les deux cas, la modélisation se base sur les opérations manipulant le type d'objet que l'on souhaite modéliser. L'exemple classique est celui de la pile d'éléments d'un type donné (pour lequel nous supposons résolu le problème de la conversion). Une pile est une structure dynamique dans laquelle on enregistre des données. La caractéristique de la pile est son mode d'accès, qui est de la forme "dernier entré, premier sorti". Plus précisément, on agit sur le domaine des piles au moyen des opérateurs suivants. L'opérateur sans argument *new* crée une pile vide. L'opérateur *empty?* appliqué à une pile p, renvoie *vrai* si p est vide et *faux* sinon. Si a est un élément, $push(p, a)$ est la pile obtenue en faisant entrer a dans p. Si p est non vide, $pop(p)$ désigne la pile après que l'on ait enlevé un élément (nécessairement le dernier entré), noté $top(p)$. Le mode d'accès se formalise au moyen des axiomes suivants.

$$
\begin{aligned}
&empty?(new)\\
&\neg\ empty?(push(p, a))\\
&pop(push(p, a)) = p\\
&top(push(p, a)) = a\\
&empty?(p)\ \vee\ push(pop(p), top(p)) = p
\end{aligned}
$$

Il est habituel de réaliser le type abstrait "pile" par le type concret "tableau". Dans ce contexte, nous voyons un tableau t comme une structure de cellules de mémoire notées $t[r]$, pour $r = 1, \ldots, n$, où n est une constante positive fixée, appelée *taille* du tableau t. Nous ne rappelons pas ici les opérations de base sur un tableau.

Une pile p sera réalisée par un tableau t et un entier naturel i, appelé index. Créer une pile p revient à créer un tableau t et un index i, dont la valeur est 0. A partir de là, les opérations sur la pile p sont réalisées par des opérations sur le tableau t et sur l'index i, en accord avec la table ci-après.

$empty?(p)$	$i = 0$
$push(p, a)$	$i := i + 1;\ t[i] := a$
$pop(p)$	$i := i - 1$
$top(p)$	$t[i]$

Le lien entre le type abstrait et sa réalisation "impérative" sera étudié au chapitre suivant. On verra que cette réalisation respecte les axiomes, tant que l'index est maintenu entre 0 et n; la pile correspond alors au sous-tableau $t[1 : i]$.

La technique constructive sera utilisée pour ériger en ensembles les couples et les nombres, donc aussi les relations et fonctions numériques.[23] Dans ce contexte, le type ensemble est le type "concret" dans lequel sont représentés les types "abstraits" (couples, nombres, etc.).

2.3.2. *Représentation ensembliste d'objets mathématiques*

Pour que l'on puisse noter et étudier des ensembles particuliers dans le cadre strict de la théorie des ensembles, il est nécessaire que les éléments de ces ensembles soient eux-mêmes des ensembles; bien plus, les éléments en question doivent pouvoir être notés au moyen du langage, très sobre, de la théorie. Ceci suffit pour introduire l'ensemble vide

$$\emptyset =_{def} \{x : x \neq x\}.$$

L'ensemble vide est aussi caractérisé par la formule $\forall y\,(y \notin \emptyset)$.

Jusqu'ici, nous avons évité l'introduction d'objets ne possédant pas la qualité d'ensemble, sauf en une occasion. L'objet x est un couple si des objets y et z existent tels que $x = (y, z)$. Pour que les couples soient des ensembles, il suffit de convenir que si y et z sont des ensembles, alors le couple (y, z) est un ensemble. Toutefois, nous souhaitons introduire aussi peu de constructeurs primitifs que possible dans la théorie, et donc rattacher l'expression (y, z) aux expressions déjà introduites. Pour ce faire, il suffit de formaliser l'idée intuitive de couple. Etant donné la paire $\{y, z\}$, on peut former les couples (y, z) et (z, y). Ces couples sont a priori distincts. Il faut préciser quel est le premier élément (l'origine) du couple. Ceci conduit à la définition

$$(a, b) =_{def} \{\{a, b\}, \{a\}\}.$$

[23] La plupart des objets mathématiques, étant modélisables par des fonctions numériques, reçoivent aussi la qualité d'ensemble.

L'objet abstrait (a, b) est donc représenté par l'objet concret $\{\{a, b\}, \{a\}\}$. Moyennant cette convention, tous les objets que nous avons définis à partir d'ensembles sont eux-mêmes des ensembles.

L'approche constructive suggère la possibilité d'étudier les objets mathématiques dans le cadre de la théorie des ensembles, à condition de définir ces objets comme des ensembles. Il peut cependant paraître abusif de définir des objets mathématiques comme des ensembles. D'une part, on peut penser que les objets mathématiques existent par eux-mêmes, ce qui interdirait de leur attribuer arbitrairement la qualité d'ensemble; d'autre part, même si l'on considère que les objets mathématiques n'existent que dans la mesure où on les crée, il peut paraître contraire à l'intuition de les réduire à un type commun. On peut effectivement admettre, sans devoir approfondir la question philosophique de leur nature, que les objets mathématiques les plus courants ont acquis une existence suffisamment tangible pour que la plupart des gens les perçoivent de manière semblable. Ainsi en est-il de la notion de nombre naturel, et des notions de point et de droite du plan. C'est aussi le cas pour les concepts de fonction et de relation. Cependant, dès que l'on examine de plus près le contenu intuitif de ces objets familiers, on observe que leur nature reste cachée et, de plus, qu'elle importe peu en pratique. Ce qui est important est plutôt la manière dont les objets sont utilisés. Par exemple, les nombres sont, par nature, susceptibles d'être additionnés. Le fait de savoir que la somme de 2 et 3 est 5 n'apporte pourtant guère d'information sur la nature même des nombres.

En géométrie, des points et des droites déterminent d'autres points et d'autres droites, par divers mécanismes, sans que l'on puisse décider quelle notion, de droite ou de point, est primitive par rapport à l'autre. Bien plus, une réponse tranchée à ce genre de question peut se révéler nuisible. Si on considère que la notion de point est primitive et celle de droite dérivée ("la droite est un ensemble infini de points"), on s'expose à ne pas voir la dualité entre ces notions, pourtant si importante en géométrie projective. En fait, la mathématique "combine" des objets, mais ne les "définit" pas.

Ceci autorise sans restriction la modélisation des objets mathématiques par d'autres objets. L'essentiel est que les modes de combinaison des objets soient respectés. On peut représenter les nombres par des ensembles pour peu que les opérations arithmétiques puissent être correctement modélisées dans ce contexte. Il en va de même des notions de la géométrie et des autres branches des mathématiques. En fait, la représentation des nombres par des ensembles n'est pas seulement licite, elle est aussi avantageuse. D'une part, elle permet de mieux unifier les diverses notions de nombres (entiers, rationnels, réels, etc.) et, d'autre part, elle permet de mieux comprendre les raisons qui ont conduit à l'introduction de ces notions. Enfin, la modélisation ensembliste des nombres a permis une meilleure compréhension de ce concept. Tant le comptage simple, "cardinal", que le comptage ordonné, "ordinal", se généralisent de façon non triviale aux ensembles infinis.

2.3.3. *Nombres naturels*

La notion mathématique la plus élémentaire est sans doute celle de nombre; il est donc indiqué de commencer par modéliser les nombres. Avant de chercher à définir des ensembles correspondant aux nombres, et des opérations correspondant aux opérations arithmétiques, il convient de s'interroger sur le contenu intuitif de la notion de nombre. Au départ, on se limite à la variété la plus simple, celle des nombres naturels.

Bien évidemment, les nombres naturels servent à *compter* et à *ordonner*. Dire d'une collection (au sens intuitif) qu'elle comporte cinq objets, c'est mettre en évidence une caractéristique de cette collection; une correspondance est établie entre cette collection et toute entité comportant cinq objets. Si l'on tient absolument à "définir" la notion "cinq", on pourra lui associer la collection de toutes les collections comportant cinq objets. Il sera toutefois plus commode d'attribuer à "cinq" un ensemble particulier, dénoté 5, qui comportera naturellement cinq éléments.

Si l'on s'intéresse à la seconde utilité des nombres naturels, c'est-à-dire l'ordonnancement, ou l'identification du rang, on arrive à des conclusions analogues. Désigner un roi de France par le terme "Louis XIV" signifie que ce roi s'appelle Louis, mais aussi que ce roi occupe un rang précis dans la liste chronologique des rois de France prénommés Louis. Cela nous conduira à imposer que la représentation ensembliste des nombres naturels soit bien ordonnée (§ 2.2.7); ce concept est intuitif: une file d'attente est bien ordonnée si, au sein de cette file, tout groupe de personnes peut déterminer qui d'entre elles est la première.

La seule représentation envisageable pour le nombre 0 est l'ensemble vide $\emptyset$, premier terme défini dans la théorie des ensembles. Le nombre 1 doit être un *singleton*, c'est-à-dire un ensemble comportant un seul élément. Cet élément sera lui-même un ensemble; il est naturel de choisir l'ensemble le "plus simple", soit 0, et d'obtenir ainsi l'ensemble $1 = \{0\}$. On continue cette démarche, pour adopter successivement

$$1 = \{0\},$$
$$2 = \{0,1\} = \{0,\{0\}\},$$
$$3 = \{0,1,2\} = \{0,\{0\},\{0,\{0\}\}\}$$

et ainsi de suite.[24]

[24] On devine déjà que disposer de l'ensemble vide comme seul élément fondamental n'est pas gênant. Ecrire $3 = \{0,\{0\},\{0,\{0\}\}\}$ n'est pas très attrayant, mais l'égalité $3 = \{0,1,2\}$ semble plus naturelle. Une autre possibilité de définition des naturels consiste à identifier le nombre naturel 3 à la classe de tous les ensembles comportant trois éléments. Pour éviter que cette seconde définition ne soit circulaire, il faudrait exhiber au moins un ensemble comportant trois éléments, ce qui semble nécessiter la première définition. On préférera donc la première définition, d'autant que la seconde donne lieu à une autre difficulté; on verra plus loin que, dans le cadre de la théorie axiomatique, la classe des triplets n'est pas un ensemble.

Ceci se généralise immédiatement, et le naturel n', successeur du naturel n, sera défini de manière à vérifier l'égalité

$$n' = \{0, 1, \ldots, n\}.$$

Cette approche aura l'avantage d'identifier la relation $<$ sur les naturels à la relation $\in$ sur les ensembles représentant les naturels. On voit immédiatement que si n est un naturel, alors le *successeur* n' de n est $n \cup \{n\}$. De même, si m et n sont des naturels et si $m \in n$, alors $m \subset n$.

Il est clair qu'un raisonnement par récurrence (nous ne remettons pas en question ici la validité de ce type de raisonnement)[26] est nécessaire pour vérifier le bien-fondé de la définition. Cela n'a rien d'étonnant, dans la mesure où le phénomène de récurrence est inséparable de celui de nombre naturel.[27]

On voit que la définition des naturels rend compte aussi bien de l'aspect ordinal que de l'aspect cardinal des naturels.

Les concepts de base de l'arithmétique des naturels sont le prédicat monadique *zerop* et la fonction monadique *add*1, que l'on introduit comme suit :

$$zerop(x) =_{def} x = 0; \quad add1(x) =_{def} x \cup \{x\}.$$

Ce prédicat et cette fonction sont définis pour tout ensemble; c'est indispensable dans la mesure où nous ne disposons pas actuellement d'un prédicat $Nat(x)$ qui permettrait de discriminer les naturels et les autres ensembles. Cette imperfection devra être corrigée, car elle empêche des déductions intéressantes. Par exemple, l'énoncé $\forall x \forall y [x \in y \Rightarrow x \subset y]$ n'est pas un théorème, mais l'énoncé

$$\forall x \forall y [(Nat(x) \wedge Nat(y) \wedge x \in y) \Rightarrow x \subset y]$$

doit en être un. La démonstration de ce "théorème provisoire" nécessite que le prédicat *Nat* soit correctement défini. Une "définition" est :

$$\forall x [Nat(x) \equiv (x = 0 \vee \exists y [Nat(y) \wedge x = add1(y)])].$$

Malheureusement, cet énoncé est récursif et, comme on l'a vu au chapitre I.9, ce type de définition n'est pas acceptable dans le contexte des théories du premier ordre. On contourne l'obstacle en remplaçant $Nat(x)$ par $(x \in \mathbf{N})$ et en définissant l'ensemble $\mathbf{N}$ des naturels de manière non récursive. On introduit d'abord le prédicat monadique *Ind* (pour "inductif") :

$$Ind(a) =_{def} [0 \in a \wedge \forall x (x \in a \Rightarrow add1(x) \in a)].$$

Un ensemble est *inductif* s'il contient 0 et le successeur de chacun de ses

[26] La justification du principe de récurrence est du ressort de la philosophie. La logique mathématique peut seulement le *formaliser*, ainsi que cela a été fait au paragraphe I.9.3.4.

[27] Le problème de la définition des naturels est analogue à celui de la définition des multiplets. Le nombre 3 et la notion de triplet se définissent aisément, sans récursivité; cette possibilité subsiste pour tout nombre entier naturel *fixé* mais la définition de l'*ensemble* des naturels ou des multiplets rend apparemment nécessaire le recours à la récursivité.

éléments. On vérifie aisément que l'intersection d'une classe[28] d'ensembles inductifs est un ensemble inductif. En effet, si x appartient à l'intersection d'une famille d'ensembles inductifs, il appartient à tous les membres de cette famille; il en va de même de son successeur $add1(x)$, qui appartient donc aussi à l'intersection.[29] On peut alors définir $\mathbf{N}$ comme le plus petit ensemble inductif, c'est-à-dire comme l'intersection de tous les ensembles inductifs:

$$\mathbf{N} =_{def} \bigcap \{x : Ind(x)\}.$$

A ce stade, nous ne pouvons toujours pas prouver le théorème provisoire!

Il peut paraître vain de poursuivre l'étude d'un concept défini de manière aussi indirecte. Ce n'est pas le cas: nous espérons au contraire qu'une meilleure connaissance des naturels permettra de mieux exploiter la définition de $\mathbf{N}$. D'ailleurs, la définition de $\mathbf{N}$ comme l'intersection de la famille des ensembles inductifs présente, outre le fait d'être non récursive, un avantage essentiel. D'un point de vue philosophique, cette définition est assez intuitive, et donc plus directe qu'il n'y paraît à première vue: elle exprime que l'ensemble $\mathbf{N}$ contient 0 et ses successeurs, et rien de plus.[30]

Puisque les axiomes de Peano constituent la base théorique la plus largement acceptée de l'arithmétique, il convient de vérifier que l'ensemble $\mathbf{N}$ introduit ici respecte ces axiomes.[31] Considérons d'abord les énoncés correspondant aux premiers axiomes de Peano:

$$\forall x\,[zerop(x) \Rightarrow x \in \mathbf{N}],$$
$$\forall x\,[x \in \mathbf{N} \Rightarrow \neg zerop(add1(x))],$$
$$\forall x \forall y\,[(x \in \mathbf{N} \wedge y \in \mathbf{N}) \Rightarrow (add1(x) = add1(y) \Rightarrow x = y)].$$

Le premier énoncé est un théorème immédiat, de même que le deuxième puisque $x \in add1(x)$, tandis que $x \notin 0$. La négation du troisième énoncé conduit au développement suivant, où x et y sont des naturels quelconques:

---

[28] Dans le cadre de la théorie axiomatique, la collection de tous les ensembles inductifs est une classe, mais pas un ensemble.

[29] Rappelons que $\bigcap \emptyset = V$. Dans la mesure ou $add1(x)$ est défini pour tout ensemble x, la classe universelle V est inductive.

[30] Dans le même ordre d'idée, soulignons que si l'idée d'ensemble inductif est utilisée sans précaution, elle peut conduire à une définition de $\mathbf{N}$ qui reste récursive; on pourrait écrire par exemple

$$\mathbf{N} = \{n : n \in \mathbf{N} \vee \exists m\,(n = add1(m) \wedge m \in \mathbf{N})\}.$$

Notre démarche a précisément été de "dérécursiver" et formaliser cette définition (ce qui ne nous empêchera pas d'utiliser, en cas de besoin, l'égalité ci-dessus, qui découle de la définition non récursive que nous avons adoptée).

[31] Les énoncés de Peano seront donc des *théorèmes* de la théorie des ensembles alors qu'ils sont des *axiomes*, ou *postulats*, de la théorie des nombres. Ceci montre que la théorie des ensembles "contient" la théorie des nombres. Plus précisément, tout théorème de la théorie des nombres sera (après traduction dans le langage de la théorie des ensembles), un théorème de la théorie des ensembles.

1.	$add1(x) = add1(y)$	hypothèse
2.	$add1(x) \subset add1(y)$	1, extensionnalité
3.	$x \cup \{x\} \subset y \cup \{y\}$	2, définition de $add1$
4.	$x \neq y$	hypothèse
5.	$\{x\} \cap \{y\} = \emptyset$	4
6.	$\{x\} \subset y \cup \{y\}$	3
7.	$\{x\} \subset y$	5, 6
8.	$x \in y$	7
9.	$x \subset y$	8, théorème provisoire
10.	$y \subset x$	permutation des rôles de x et y dans 1...9
11.	$x = y$	9, 10, extensionnalité
12.	*false*	4, 11

La réciproque $add1^{-1}$ de la fonction $add1$ est une restriction de la fonction $sub1$ dont voici la définition :

$$sub1(x) =_{def} \bigcup x .$$

Remarque. Les objets $add1(x)$ et x', d'une part, et $sub1(x)$ et $\bigcup x$, d'autre part, sont identiques. On convient cependant de n'utiliser les notations $add1$ et $sub1$ que pour les ensembles représentant des nombres naturels, alors que $x' = x \cup \{x\}$ et $\bigcup x$ sont définis pour tout ensemble x. D'autre part, on écrit souvent $x+1$ et $x-1$ pour $add1(x)$ et $sub1(x)$, même dans un contexte où l'addition générale $x+y$ n'a pas été formellement définie.

On observe que $\bigcup\{0, 1, \ldots, n\} = \bigcup\{n\} = n$; de plus, $(x \in y \wedge y \in n)$ implique $x \in sub1(n)$. Plus formellement, on a les théorèmes suivants :

$$sub1(0) = 0 ,$$
$$\forall x\,[x \in \mathbf{N} \Rightarrow sub1(add1(x)) = x] ,$$
$$\forall x\,[(x \in \mathbf{N} \wedge \neg zerop(x)) \Rightarrow add1(sub1(x)) = x] ,$$
$$\forall x \forall y \forall z\,[(z \in y \wedge y \in x \wedge x \in \mathbf{N}) \Rightarrow (\neg zerop(x) \wedge z \in sub1(x))] .$$

Le lecteur pourra observer la similitude existant entre la structure de pile et la structure de nombre naturel; la correspondance entre les deux notions est résumée dans la table ci-dessous.

new	0
empty?	*zerop*
push	*add1*
pop	*sub1*

Les autres opérations arithmétiques se définissent facilement. L'addition et la multiplication sont introduites par les énoncés suivants, dans lesquels x et y sont des naturels.

$$add(x, 0) =_{def} x ,$$
$$add(x, add1(y)) =_{def} add1(add(x, y)) ,$$
$$mult(x, 0) =_{def} 0 ,$$
$$mult(x, add1(y)) =_{def} add(x, mult(x, y)) .$$

Ces "définitions" sont récursives et donc inacceptables ici, dans le cadre d'une théorie du premier ordre.[32] Nous nous heurtons pour la quatrième fois à l'obstacle de la récursivité, déjà rencontré à propos de la notion de multiplet (laissée en suspens, § 2.2.4), puis de la preuve du théorème 2.1, puis de la définition de l'ensemble **N** (pour lequel nous avons pu donner une définition alternative, non récursive). Ceci semble indiquer que la notion de récursivité est difficilement évitable en théorie des ensembles; celle-ci visant à la modélisation des phénomènes mathématiques les plus généraux, cela n'est pas vraiment surprenant. Il est donc utile de présenter ce type de définition dans un contexte assez général.

2.3.4. *Définitions récursives*

2.3.4.1. *Introduction*

Une définition, au sens habituel du terme, est *récursive* si l'objet défini est évoqué dans le texte définissant. Une définition non récursive d'un concept Φ est de la forme "Φ est Ψ", où Ψ est un texte n'évoquant que des concepts déjà définis. La définition est récursive si Ψ évoque en outre le concept Φ lui-même. A priori, les définitions récursives sont des cercles vicieux et il convient de les éviter. On peut cependant admettre les définitions récursives dans certains cas. En particulier, la définition récursive de la factorielle n'est pas contestable; celle des coefficients binomiaux, basée sur le triangle de Pascal, est admise depuis des siècles. Bien plus, certaines notions, telle la suite de Fibonacci, n'admettent de définition élémentaire que récursive. Ces exemples, pour particuliers qu'ils soient, sont trop fréquents et importants pour être ignorés, et l'on souhaiterait disposer d'une méthode relativement générale permettant de déterminer si une définition récursive est acceptable ou non.

Avant d'aborder ce problème, notons aussi que la reconnaissance même du caractère récursif d'une définition requiert une certaine prudence. Dans l'exemple du multiplet ordonné, le premier que nous avons rencontré, la récursivité n'intervient pas immédiatement. Par exemple, la définition du quadruplet ordonné n'est pas récursive; en effet, l'expression définissante

[32] Une technique non récursive pour introduire l'addition consisterait à décrire la "table d'addition" des naturels. Cependant, cette table est nécessairement infinie, ce qui pose un problème pratique évident. Il y a également un problème théorique: nous avons déjà rencontré plusieurs situations où l'extension au cas infini d'un concept initialement prévu pour le cas fini s'avère délicate ou impossible. Une autre idée de définition non récursive est la suivante. Si l'ensemble a est en bijection (§ 2.2.3) avec le naturel m, si l'ensemble b est en bijection avec le naturel n et si a et b sont disjoints ($a \cap b = \emptyset$), alors il existe un et un seul naturel en bijection avec la réunion $a \cup b$, et ce naturel est par définition la somme $m + n$. Il serait néanmoins préférable de justifier l'usage, en théorie des ensembles, d'une certaine forme de récursivité; en effet, le remplacement d'une définition récursive par une définition non récursive équivalente n'est pas toujours commode.

comporte un couple et un triplet (dont la définition repose seulement sur celle de couple), mais pas de quadruplet. On peut donc, de manière tout à fait licite, définir successivement (et dans cet ordre !) les notions de triplet, de quadruplet, de quintuplet, etc. On pourrait également, au moyen d'un raisonnement par récurrence, justifier le bien-fondé de la notion de n-uplet ordonné, quelque soit le naturel n. Cela permettrait peut-être de définir ensuite la notion de multiplet : a est un multiplet s'il existe un nombre naturel n tel que a est un n-uplet.[33] La définition "générique" de n-uplet que nous avons introduite est peut-être valide (au sens d'une éventuelle théorie de la définition) mais elle est indubitablement récursive, car le symbole défini, à savoir la paire de parenthèses entourant un nombre quelconque de termes séparés par des virgules, apparaît dans l'expression définissante. Avant d'approfondir le concept de définition récursive, il convient d'éclaircir la notion même de définition.

2.3.4.2. *Qu'est-ce qu'une définition ?*

Définir la notion de définition n'est pas possible en toute généralité, mais on peut quand même souligner deux points. Tout d'abord, un critère ne peut être assimilé à une définition que s'il permet de distinguer le concept défini de tout autre concept; l'idée d'unicité est inséparable de celle de définition.[34] Ensuite, si l'expression "Φ est Ψ" est une définition, il doit être possible, sous certaines conditions peu restrictives, de substituer Ψ à Φ; la substitution inverse est souvent licite, elle aussi.[35]

Si nous nous restreignons à la définition d'objets mathématiques (est-ce réellement une restriction ?) on souhaitera pouvoir affirmer, au départ de toute définition, l'égalité du concept défini et de l'expression définissante; l'égalité $\Phi = \Psi$ devra donc être valide.[36]

2.3.4.3. *Définitions récursives et intuition*

Tout naturellement, nous avons tendance à accepter une définition si nous voyons un moyen de la mettre en œuvre. La définition récursive des coefficients

[33] En fait, rien ne prouve qu'une notion puisse être définie en termes d'une *infinité* de notions prédéfinies; nous n'exploiterons pas ce type de définition.

[34] Il ne faut pas confondre l'unicité du concept, et la multiplicité éventuelle des instances du concept. Par exemple, la notion de parité est unique, puisqu'elle est définissable. Cela ne signifie pas qu'il n'y a qu'un nombre pair mais plutôt que, pour tout nombre x, la question "x est-il pair" n'admet qu'une réponse correcte. Autrement dit, l'ensemble des nombres pairs est unique.

[35] Une définition est néanmoins foncièrement asymétrique (l'expression "Ψ est Φ" ne sera généralement pas une définition). Les conditions dans lesquelles la substitution est possible ne sont pas toujours faciles à préciser. On notera quand même que la substitution est normalement interdite dans la définition elle-même; "Φ est Φ" n'est pas une définition.

[36] C'est bien le cas pour les mécanismes (non récursifs) de définition en extension et en compréhension.

binomiaux illustre bien ce point. On a

$$\begin{aligned} C_n^0 &=_{def} 1, \text{ pour tout } n \in \mathbf{N}, \\ C_n^n &=_{def} 1, \text{ pour tout } n \in \mathbf{N}, \\ C_n^p &=_{def} C_{n-1}^p + C_{n-1}^{p-1}, \text{ pour tous } n \in \mathbf{N} \text{ et } p \in \{1, \ldots, n-1\}. \end{aligned}$$

(Rappelons que C_n^p est le nombre de sous-ensembles de p éléments d'un ensemble de n éléments.) Formellement, ce texte est assez complexe; il est cependant facile à utiliser. On calculera C_4^2, par exemple, au moyen d'une série de réductions:

$$\begin{aligned} C_4^2 &= C_3^2 + C_3^1 = (C_2^2 + C_2^1) + (C_2^1 + C_2^0) \\ &= (C_2^2 + (C_1^1 + C_1^0)) + ((C_1^1 + C_1^0) + C_2^0) \\ &= (1 + (1 + 1)) + ((1 + 1) + 1) = 6. \end{aligned}$$

D'autre part, la définition suggère une tabulation naturelle de ces coefficients, sous la forme du triangle de Pascal.

La généralisation est simple; une forme $h(x)$, où x est un paramètre ou un argument (simple ou multiple) peut être définie par rapport à elle-même à condition que, pour toute occurrence $h(e_x)$ apparaissant dans l'expression définissante, la valeur de l'argument e_x soit "plus simple" que la valeur de x. Si l'argument est un nombre naturel, on traduira "plus simple" en "plus petit". Si l'argument est un entier relatif, cette traduction n'est plus valable, car un entier relatif, au contraire d'un entier naturel, admet une infinité de minorants.

Intuitivement, une définition récursive est acceptable si elle suggère un processus d'évaluation ou de réduction borné. Cette idée fondamentale est simple même si, dans certain cas, le caractère borné du processus de réduction n'est pas facile à déterminer.

2.3.4.4. *Définitions récursives et équations*

Une "égalité définissante" n'est rien d'autre qu'une équation particulière; elle est soumise, en apparence, à deux conditions. L'inconnue (objet défini) apparaît isolément comme membre de gauche, et l'équation admet une solution unique. Une troisième condition, essentielle mais trop souvent implicite, est l'existence d'un "référentiel" ou domaine de l'équation. L'équation

$$x = \frac{1}{2}\left(x + \frac{A}{x}\right),$$

où A est un réel positif, définit le réel positif $\sqrt{A}\ldots$ si on précise que le domaine de référence est $\mathbf{R}^+$. Si ce domaine est $\mathbf{R}$, l'équation admet deux solutions et n'est donc pas définissante. Selon ce point de vue, les équations définissent des solutions, et seules les équations triviales[37] correspondent à des définitions non

[37] dont l'inconnue est le premier membre et n'apparaît pas dans le second.

récursives; les définitions récursives sont donc la règle, et les définitions non récursives sont l'exception. Le remplacement d'une définition récursive par une autre, non récursive, est un problème voisin de la résolution d'équation; sa complexité est donc très variable.[38]

Nous ne développerons pas plus avant ces considérations générales sur la récursivité. Par contre, il est intéressant de répertorier les attitudes possibles du logicien formel devant introduire un nouveau concept pour lequel une définition récursive est envisageable.

2.3.4.5. *Récursivité et logiques formelles*

Une attitude extrême en logique consiste à refuser, purement et simplement, toute forme de récursivité. Les concepts pour lesquels une définition non récursive n'est pas connue ne feront pas l'objet d'une définition. Ils ne seront pas ignorés pour autant : des axiomes se rapportant à un tel concept peuvent pallier l'absence de définition. Cette attitude est celle de théorie axiomatique des nombres, et celle de la logique du premier ordre en général. Ce point de vue élémentaire a de grands avantages; en particulier, les théories du premier ordre sont généralement plus simples que les autres, leurs propriétés sont plus faciles à déterminer, et leur mise en œuvre, éventuellement informatique, peut rester élémentaire. Il n'y a évidemment pas de miracle. Même formalisée dans ce cadre, l'arithmétique, par exemple, reste fondamentalement difficile. De plus, le caractère élémentaire des théories du premier ordre s'accompagne d'une rigidité gênante.[39]

L'attitude opposée consiste à admettre toute forme de récursivité. Etant donné un référentiel R et une équation du type $x = E_x$ sur ce référentiel, on "postule" l'existence et l'unicité d'une solution dans R, que l'on note par exemple $\mu(x : E_x)$. On peut sans inconsistance postuler l'unicité si le référentiel R est bien ordonné; $\mu(x : E_x)$ désigne conventionnellement la plus petite solution de l'équation; de même, on convient qu'un élément particulier de R est solution de toutes les équations n'admettant pas d'autres solutions.

---

[38] Cette analogie, qui banalise la récursivité, ne doit pas être poussée trop loin. Une équation n'existe pas tant que son référentiel (ensemble de valeurs associé à l'inconnue) n'a pas été défini. Ce référentiel, s'il est lui-même défini récursivement, implique l'existence d'un autre référentiel prédéfini. Il doit y avoir au départ un ou plusieurs objets non définis, ou définis non récursivement. En outre, les équations triviales sont nettement plus simples que les autres; il ne faut donc pas utiliser la récursivité sans raison.

[39] Ajouter un nouveau concept peut modifier radicalement une théorie du premier ordre, même si le nouveau concept se définit récursivement en fonction des anciens. Par exemple, la théorie de Presburger comporte les symboles 0, 1 et +, et ses axiomes sont ceux de Peano qui n'évoquent pas la multiplication. La théorie est complète et décidable, ce qui, vu les résultats célèbres de Gödel, n'est plus le cas si la multiplication et ses axiomes sont ajoutés. En résumé, le refus de la récursivité en logique du premier ordre réduit le pouvoir de définition à un mécanisme d'abréviation.

Cette attitude est adoptée dans diverses logiques plus puissantes que la logique classique, et notamment dans le μ-calcul et ses variantes. Plus une logique est puissante (au point de vue de l'expressivité) moins elle a de propriétés (décidabilité, ...) et plus sa mise en œuvre est difficile. Des exceptions existent, ce qui nous incitera à reconsidérer cette attitude au chapitre 5.

On peut aussi envisager une attitude moyenne, et utiliser la récursivité à condition de prouver qu'elle n'augmente pas l'expressivité *théorique* du cadre dans lequel on travaille. C'est ce que nous ferons en théorie des ensembles.

2.3.4.6. *Récursivité et théorie des ensembles*

La "dérécursivation" des définitions, en logique du premier ordre, est coûteuse et aléatoire. Le cas de l'ensemble des naturels est une heureuse exception; la notion-clef d'ensemble inductif, quoique non triviale, reste élémentaire. Dans le cadre des théories du premier ordre, le remplacement des définitions récursives par des définitions non récursives n'est pas une technique prometteuse.[40] Par contre, et c'est un avantage décisif de la théorie des ensembles, disposer de l'ensemble $\mathbf{N}$ permettra d'introduire une certaine forme de récursivité dans les mécanismes de définition, sans se soucier explicitement de l'existence d'équivalents non récursifs.

Rappelons encore que les symboles extralogiques d'une théorie du premier ordre sont d'habitude fixés a priori; l'introduction d'un nouveau symbole et des axiomes qui le concernent donne lieu à une nouvelle théorie, parfois fort différente de la théorie initiale. Il est cependant admis (et en pratique, c'est inévitable) d'utiliser de nouveaux symboles pour abréger des expressions licites d'une théorie. On peut par exemple, dans n'importe quelle théorie, abréger la formule $\neg(x = y)$ en $x \neq y$; dans la théorie de l'ordre, la formule $x < y$ abrège la formule $x \leq y \wedge x \neq y$. La théorie des ensembles se distingue des autres théories du premier ordre par l'extrême fécondité des mécanismes abréviatifs. Au départ du symbole d'appartenance et de la logique, on peut définir une série de constantes individuelles (désignant des ensembles particuliers, comme $\emptyset$ et $\mathbf{N}$), prédicatives et fonctionnelles (réunion, produit, etc.). En théorie cependant, les symboles introduits ne sont pas indispensables; leur rôle est simplement d'alléger et de clarifier les notations. On peut toujours éliminer les symboles dérivés; voici un exemple d'élimination :

$$1 \subset 2,$$
$$\forall x\,[x \in 1 \Rightarrow x \in 2],$$
$$\forall x\,[x \in \{0\} \Rightarrow x \in \{0, \{0\}\}],$$

[40] En programmation, au contraire, des techniques simples permettent de remplacer un programme récursif par un programme non récursif utilisant une pile.

$$\forall x\,[x = 0 \Rightarrow (x = 0 \vee x = \{0\})]\,,$$
$$\forall x\,[x = 0 \Rightarrow (x = 0 \vee \forall y\,[y \in x \Rightarrow y = 0])]\,,$$
$$\ldots \qquad \ldots \qquad \ldots \qquad \ldots$$

Pour terminer, il faut encore remplacer "$x = 0$" par "$\forall z\,(z \in x \Rightarrow \neg(z = z))$" (deux fois) et éliminer de manière analogue "$y = 0$". Signalons encore que le symbole d'égalité peut être éliminé aussi; on peut remplacer "$a = b$" par "$\forall u\,(u \in a \equiv u \in b)$".

On peut introduire une nouvelle notation pour désigner tout objet dont l'existence et l'unicité ont été démontrées. Par exemple, dans le cadre de la théorie des groupes, chaque élément admet un et un seul inverse; on peut donc désigner par x^{-1} l'unique inverse de x. Dans le cadre de la théorie des ensembles, on peut représenter par $\{a, b, c\}$ l'unique ensemble dont les éléments sont les ensembles a, b et c; on peut définir aussi $1 = \{0\}$.[41] En ce qui concerne la théorie des ensembles, nous avons déjà admis l'introduction d'une notation nouvelle pour tout ensemble dont l'existence et l'unicité a été prouvée; l'expression $\{a, b, c\}$ a été justifiée de cette manière. Il est donc naturel d'admettre une notation Φ introduite par une définition récursive "Φ est Ψ" à condition que l'on prouve l'existence et l'unicité de la solution de l'équation associée $\Phi = \Psi$.[42]

Considérons à nouveau le cas de l'addition. La "définition" du paragraphe 2.3.3 est récursive; pour mettre en évidence le fait que l'objet "défini" est la fonction *add*, on peut utiliser la notation λ:

$$add \;=\; \lambda(x, y)\,[\text{si } y = 0 \text{ alors } x \text{ sinon } add1(add(x, sub1(y)))]\,.$$

Dans la mesure où l'introduction du symbole *add* dans la signature de la théorie est exclue,[43] nous devons le définir, c'est-à-dire justifier la définition récursive. Il faut prouver que les égalités du paragraphe 2.3.3 sont vraies si

---

[41] Ce type de technique abréviative doit cependant s'utiliser avec prudence. Il est parfois indiqué de récrire les axiomes de la théorie en fonction de la nouvelle notation. Le cas de la théorie des groupes est instructif à cet égard. Si une notation explicite est introduite pour désigner l'inverse d'un élément, il est souhaitable de remplacer l'axiome $\forall x\,\exists y\,(x.y = e = y.x)$ par l'axiome $\forall x\,(x.x^{-1} = e = x^{-1}.x)$. (L'ancien axiome devient un théorème de la nouvelle théorie.) Cette modification n'est pas gratuite; en effet, les axiomes de la nouvelle version ne comportent plus de quantification existentielle, ce qui confère à la nouvelle théorie des propriétés intéressantes (mais ne change rien aux *modèles* de la théorie, évidemment).

[42] On notera aussi, sur le plan technique, que l'usage d'une notation Φ pour représenter la fonction (de domaine a et de codomaine inclus dans b) définie récursivement par le texte "Φ est $\Psi(\Phi)$" préserve la possibilité théorique d'élimination. Soit $\mathcal{A}(\Phi)$, une formule dans laquelle Φ a des occurrences libres. On peut éliminer le symbole Φ et donc construire une formule équivalente à $\mathcal{A}(\Phi)$ et ne contenant pas Φ; cette formule sera

$$\forall f\,[(dom(f) = a \wedge cod(f) \subset b \wedge f = \Psi(f)) \Rightarrow \mathcal{A}(f)]\,.$$

En effet, si l'existence et l'unicité ont été prouvées, l'antécédent de l'implication est vrai pour $f = \Phi$ et faux pour tout autre objet.

[43] On veut précisément éviter tout changement de théorie: la théorie des ensembles a l'ambition de modéliser tous les objets mathématiques dans un même cadre.

"*add*" désigne l'addition, mais fausses si "*add*" désigne toute autre fonction, ou un objet quelconque non fonctionnel. Cela n'est pas difficile, mais une preuve est indispensable; une preuve analogue devra être fournie lors de chaque nouvelle définition. Pour éviter cette démarche répétitive, il serait commode d'isoler un "schéma de définition", suffisamment général pour la majorité des besoins, mais suffisamment restrictif pour faire l'objet d'une preuve "générique", établissant la validité de toutes les définitions susceptibles d'être obtenues en instanciant le schéma.

2.3.4.7. *Vers un schéma de définition récursive*

A ce stade de notre étude, il semble que la récursivité soit intimement liée à l'ensemble des naturels, et qu'elle survienne à propos de lois ou de fonctions visant à associer à tout naturel un objet qui, en toute généralité, sera un ensemble (ou une classe). La notion de multiplet évoque une fonction qui, à tout naturel n associe la classe des n-uplets.[44] L'addition paraît a priori une entité plus complexe, puisqu'elle associe un objet à tout *couple* de naturels. La *technique de Curry* permet de réduire ce cas au précédent. Plus généralement, une bijection naturelle existe entre les ensembles fonctionnels $(A \times B) \to C$ et $A \to (B \to C)$. Si la fonction f est un élément du premier ensemble, sa forme de Curry f_C est définie comme suit :

$$(f_C(a))(b) = c \text{ si et seulement si } f(a,b) = c\,,$$

où a, b, c sont des éléments quelconques de A, B, C, respectivement. La technique de Curry permet de ramener les fonctions à plusieurs arguments (telle l'addition) à des fonctions à un seul argument. Le cas de l'addition est particulièrement simple. Si on note $add(m)$ la fonction qui, à tout naturel n, associe le naturel $m+n$, on a $add(m,n) = m+n = [add(m) \circ add(n)](0)$.

Nous nous intéressons donc à la définition récursive de fonctions de $\mathbf{N}$ dans A, où A est une classe quelconque. Les fonctions de ce type abondent en arithmétique, et voici trois exemples classiques :

$$x^2 = x.x\,,$$
$$0! = 1\,,\quad x > 0 \Rightarrow x! = x.(x-1)!\,,$$
$$fib(0) = 0\,,\quad fib(1) = 1\,,\quad x > 1 \Rightarrow fib(x) = fib(x-1) + fib(x-2)\,.$$

La quantification universelle de x, restreinte à $\mathbf{N}$, est implicite.
Dans les trois cas, on définit une fonction h de domaine $\mathbf{N}$ et de codomaine inclus dans $\mathbf{N}$ en donnant la valeur de $h(x)$, pour tout $x \in \mathbf{N}$. Le procédé n'est pas critiquable, à condition que l'expression définissant $h(x)$ ait elle-même été définie. C'est le cas pour la fonction carré, dont la définition n'est pas récursive, mais ce n'est plus nécessairement le cas pour les fonctions

[44] Dans le cadre de la théorie axiomatique, la classe des n-uplets n'est pas un ensemble, sauf si n est nul.

factorielle et de Fibonacci, pour lesquelles le terme $h(x)$ est défini par une expression contenant h. Tant le point de vue intuitif que le point de vue équationnel suggèrent que le "cercle vicieux apparent" est admissible parce que l'expression définissant $h(x)$ contient un nombre fini de termes de type $h(e_x)$ et que, pour chaque occurrence, e_x est plus petit, plus simple que x. Autrement dit, l'expression définissant $h(x)$ évoque exclusivement des valeurs appartenant à l'ensemble $\{h(0), h(1), \ldots, h(x-1)\}$. En termes opérationnels, le calcul de l'expression $h(x)$ exige le calcul préalable d'un certain ensemble d'expressions similaires de type $h(y)$ mais ces expressions sont toutes "plus simples" ($y < x$) et sont en nombre fini (au plus x expressions auxiliaires).

Sur base des points de vue intuitif et équationnel, ainsi que des quelques exemples rencontrés, un schéma de définition récursive à la fois simple et expressif est $h(x) = F(x, h(0), \ldots, h(x-1))$. Il permet de définir des fonctions dont le domaine est l'ensemble des nombres naturels.

2.3.4.8. *Terminologie*

Nous avons utilisé le mot "récursivité" dans son acception la plus fréquente. La récursivité est un attribut qu'une définition peut avoir ou ne pas avoir. Une inspection (attentive) de l'expression définissante permet de déterminer si une définition est ou n'est pas récursive. Une acception différente, mais encore fort proche, est employée dans l'expression "programme récursif". Un programme définit, de manière plus ou moins directe, une fonction (associant aux données du programme les résultats qu'il fournit). Dans ce contexte, un programme est une définition et peut donc être récursif.

Depuis les célèbres travaux de Gödel, Turing et Church, l'expression "fonction récursive" ne signifie plus seulement "fonction définie récursivement" mais aussi "fonction programmable". Nous montrerons plus loin que, dans un sens à préciser, l'ensemble infini des fonctions de $\mathbf{N}$ dans $\mathbf{N}$ est "beaucoup plus grand" que l'ensemble, également infini, des programmes écrits dans un langage classique, tel Fortran ou Pascal. Cela implique que l'écrasante majorité des fonctions de $\mathbf{N}$ dans $\mathbf{N}$ ne sont pas programmables dans ces langages ou, selon la terminologie de Gödel, ne sont pas récursives. Cette deuxième notion de récursivité est fort différente de la première; en particulier, la notion introduite par Gödel est une propriété intrinsèque de la fonction et ne dépend pas de la manière dont cette fonction est définie.

Le mot "récursion" est parfois employé comme synonyme de "récursivité"; plus souvent, la récursion est le processus par lequel une forme $f(x)$ est évaluée quand la fonction f est définie récursivement. (Nous avons illustré plus haut ce processus en évaluant C_4^2.)

Le mot "récurrence" se rapporte d'abord à la technique de raisonnement bien connue. L'expression "définition par récurrence" est parfois employée pour "définition récursive". Il s'agit en fait d'une définition dont la validité

est démontrée par (le principe de) récurrence. Nous verrons plus loin que la technique de définition récursive peut se généraliser aux fonctions dont le domaine est n'importe quel ensemble bien fondé; le terme "récurrence" n'est habituellement utilisé que dans le cas particulier de l'ensemble des naturels (le seul évoqué jusqu'ici). Le principe de raisonnement par récurrence fait l'objet d'une généralisation analogue; on parle alors de principe d'*induction*[45] et la récurrence s'identifie à l'induction sur les naturels.

2.3.5. *Le théorème de von Neumann*

Le schéma de définition récursive

$$h(x) = F(x, h(0), \ldots, h(x-1))$$

n'est pas acceptable tel quel, même en tant que simple égalité. Les points de suspension indiquent ici que la "fonction" F a un nombre variable d'arguments, ce qui empêche de lui attribuer un domaine. L'égalité $x = \{0, 1, \ldots, x-1\}$ permet de remédier à cela; on peut regrouper les arguments de F (sauf le premier) en l'ensemble $cod(h|_x)) = \{h(0), \ldots, h(x-1)\}$. Le schéma

$$h(x) = F(x, cod(h|_x))\,,$$

où la fonction F a maintenant deux arguments, est une égalité bien construite. En tenant compte des identités $x = \{y : y < x\} = dom(h|_x)$, valables pour tout naturel x, on peut simplifier le schéma en :

(2.1) $\quad h(x) = F(h|_x)$, pour tout $x \in \mathbf{N}$.

Ce schéma est le *schéma de von Neumann*; c'est un ensemble d'égalités, dépendant de la fonction F. La discussion du paragraphe précédent a montré que ces égalités constituaient une définition acceptable de h si une et une seule fonction h les vérifiait toutes. On a à ce propos le théorème suivant.

Théorème 2.2. Soient A une classe[46] non vide et Φ la classe des fonctions dont le domaine est un naturel et le codomaine est inclus dans A. Si F est une fonction de Φ dans A, alors il existe une et une seule fonction h de $\mathbf{N}$ dans A telle que

$$h(n) = F(h|_n)\,, \text{ pour tout } n \in \mathbf{N}\,.$$

Principe de la preuve. Il suffit de formaliser le processus d'approximations successives qui doit conduire à la fonction h. Ces approximations forment la suite de fonctions (h_n) telle que $h_0 = 0 = \emptyset$ et $h_{n+1} = h_n \cup \{(n, F(h_n))\}$.

Remarques. Rappelons encore que la définition de h_{n+1}, pour n fixé, n'est pas récursive, puisque l'expression définissante comporte h_n mais pas h_{n+1}. Notons aussi l'intervention d'un concept d'*approximation.* En effet, si le

[45] Ce mot a, lui aussi, d'autres acceptions ...

[46] Dans le cadre axiomatique, imposer que A et Φ soient des ensembles serait gênant.

schéma de von Neumann ne faisait qu'*identifier* sans ambiguïté l'objet défini, sa portée *pratique* resterait faible. Ce schéma permet aussi de construire des approximations arbitrairement précises de l'objet défini, ce qui le rend précieux en pratique.[47]

Preuve du théorème. On considère la suite de fonctions $(h_n : n \in \mathbf{N})$ telle que $h_0 = \emptyset$, $h_{n'} = h_n \cup \{(n, F(h_n))\}$, pour tout $n \in \mathbf{N}$. On a la propriété suivante: pour tout $n \in \mathbf{N}$, h_n est une fonction de domaine n et, pour tout $m < n$ (c'est-à-dire $m \in n$), on a $h_m = h_n|_m$. Ceci se démontre par l'absurde. Soit ρ le plus petit naturel pour lequel la propriété est fausse. On a $\rho \neq 0$ car $h_0 = \emptyset$ vérifie la propriété; ρ est donc successeur d'un naturel σ et on a $h_\rho = h_\sigma \cup \{(\sigma, F(h_\sigma))\}$. Comme h_σ est une fonction de domaine σ, h_ρ est une fonction de domaine $\rho = \sigma \cup \{\sigma\}$. De plus, pour tout $\tau \in \rho$, on a $h_\tau = h_\rho|_\tau$. On distingue deux cas. Si $\tau \in \sigma$, alors $h_\tau = h_\sigma|_\tau = h_\rho|_\tau$ puisque h_σ et h_ρ ne diffèrent que par un couple dont l'origine σ n'appartient pas à τ. En outre, si $\tau \notin \sigma$, on a $\tau = \sigma$ et $h_\sigma = h_\rho|_\sigma$ par construction, d'où une contradiction.
On définit maintenant $h =_{def} \bigcup\{h_n : n \in \mathbf{N}\}$. L'objet h est un ensemble de couples (relation binaire). Le domaine de la relation h est $\mathbf{N}$ puisque, pour tout $n \in \mathbf{N}$, h inclut un élément dont le domaine contient n. Cette relation est une fonction et $h(n)$ est égal à $h_p(n)$, quel que soit p tel que $n \in p$.
Il reste à montrer que h est la seule fonction pour laquelle le schéma de von Neumann est vérifié. Soit, par l'absurde, une autre solution h' et soit ρ le plus petit naturel tel que $h'(\rho) \neq h(\rho)$. On a donc $h'(\tau) = h(\tau)$ pour tout $\tau \in \rho$, c'est-à-dire $h'|_\rho = h|_\rho$. On en déduit $h'(\rho) = F(h'|_\rho) = F(h|_\rho) = h(\rho)$, d'où une contradiction.

On vérifie la validité d'une définition récursive en construisant explicitement la fonction F correspondante. Pour les fonctions factorielle et de Fibonacci, la fonction F admet pour argument toute fonction φ dont le domaine est un naturel n et dont le codomaine est inclus dans $A = \mathbf{N}$. (On appelle souvent *fonctionnelle* une fonction telle que F, dont l'argument (ou l'un des arguments) est lui-même une fonction.) Dans le cas de la factorielle, l'objet $F(\varphi)$, résultant de l'application de la fonctionnelle F à la fonction φ, sera le naturel défini par les égalités

$$F(\varphi) = 1, \text{ si } n = dom(\varphi) = 0,$$
$$F(\varphi) = n \,.\, \varphi(n-1), \text{ si } n = dom(\varphi) > 0.$$

[47] On pourrait naïvement objecter que, la mathématique étant une science exacte, un processus d'approximation est insuffisant. Il faut bien observer que la connaissance *approximative* d'un objet mathématique permet de répondre de manière *exacte* à *certaines* questions le concernant. Ce point est essentiel en informatique. Concrètement, connaître une approximation (au sens précis qui va être introduit dans la preuve du théorème) de la fonction de Fibonacci permet de déterminer exactement la valeur de $fib(n)$, pour peu que n soit inférieur à un certain nombre. Cette limite pouvant être arbitrairement reculée, on accède ainsi à une connaissance exacte et complète de la fonction *fib*.

Dans le cas de la fonction de Fibonacci, si φ est une fonction de domaine $n \in \mathbf{N}$ et de codomaine inclus dans $\mathbf{N}$, l'objet $F(\varphi)$ sera le naturel défini par les égalités

$$\begin{aligned} F(\varphi) &= 0\,, \text{ si } n = 0\,, \\ F(\varphi) &= 1\,, \text{ si } n = 1\,, \\ F(\varphi) &= \varphi(n-1) + \varphi(n-2)\,, \text{ si } n > 1\,. \end{aligned}$$

Remarque. On peut, même si cela n'a pas d'intérêt, considérer les définitions non récursives comme des cas particuliers des définitions récursives; pour la fonction carré, on aura

$$F(\varphi) = n^2\,, \text{ où } n = dom(\varphi)\,.$$

Le membre de droite ne dépend que du domaine de la fonction φ, et non des valeurs prises par cette fonction.

Les définitions récursives "pathologiques" sont à la base d'un grand nombre d'erreurs, parfois difficiles à détecter. Le moyen systématique de les éviter est de n'accepter une définition qu'après en avoir vérifié la conformité au théorème de von Neumann. La négligence la plus commune consiste à confondre une égalité de type $h(n) = F(h, n)$ avec l'égalité $h(n) = F(h|_n)$. Dans ce cas, le plus souvent, il n'existera pas de fonction h vérifiant l'égalité ou, au contraire, il en existera plusieurs, voire une infinité. Deux cas extrêmes d'équations inacceptables en tant que définitions récursives sont $h(n) = h(n)$ (toute fonction de domaine $\mathbf{N}$ est solution) et $h(n) = h(n) + 1$ (il n'y a pas de solution). Un cas intermédiaire est $h(n) = h(n+1)$; les solutions sont les fonctions constantes. Dans certains contextes, il est néanmoins souhaitable d'utiliser de telles équations. Nous reviendrons sur ce point en détail au chapitre 5.

A titre d'application supplémentaire, on peut observer que la notation $f^{(n)}(x)$, où $x \in E$ et $f \in (E \to E)$, souvent "définie" par les égalités $f^{(0)}(x) = x$ et $f^{(n+1)}(x) = f(f^{(n)}(x))$, désigne l'image de n par une certaine fonction h. Cette fonction $h : n \to f^{(n)}(x)$ est la solution du schéma de von Neumann, quand F associe à toute fonction φ, de domaine $n \in \mathbf{N}$ et de codomaine inclus dans $A = E$, la valeur x si $n = 0$ et la valeur $f(\varphi(n-1))$ si $n > 0$. On peut aussi définir directement la fonction $n \to f^{(n)}$; c'est la solution h du schéma de von Neumann, quand F associe à toute fonction φ, de domaine $n \in \mathbf{N}$ et de codomaine inclus dans $A = (E \to E)$, la fonction identité sur E si $n = 0$ et la fonction $f \circ \varphi(n-1)$ si $n > 0$. La fonction h est la fonction d'itération relative à f.

Dans la forme simple qui vient d'être introduite, le schéma de von Neumann permet de définir exclusivement des fonctions de domaine $\mathbf{N}$. On ne peut donc pas définir d'emblée l'addition, dont le domaine doit être $\mathbf{N} \times \mathbf{N}$. Par contre, si m est un naturel quelconque mais fixé, on peut définir une fonction $add(m)$, de domaine $\mathbf{N}$ et de codomaine inclus dans $\mathbf{N}$, telle que $add(m)(n) = m+n$, pour tout $n \in \mathbf{N}$. Le schéma de von Neumann s'applique, avec $A = \mathbf{N}$; la fonction F associe à toute fonction φ dont le domaine est un naturel n et le codomaine un

sous-ensemble de $\mathbf{N}$, la valeur définie par les égalités

$$F(\varphi) = m\,, \text{ si } n = 0\,,$$
$$F(\varphi) = add1(\varphi(sub1(n)))\,, \text{ si } n > 0\,.$$

La fonction $add(m)$ est donc l'unique solution h des égalités

$$h(n) = F(h|_n)\,, \text{ pour tout } n \in \mathbf{N}_0\,.$$

Il est alors immédiat de définir

$$add(m,n) = m + n \;=_{def}\; add(m)(n)\,, \text{ pour tous } m,n \in \mathbf{N}.$$

Les définitions des fonctions $add(m)$ sont récursives, mais la définition de add ne l'est pas.

Remarque. On peut aussi définir $add(m) = add1^{(m)}$

Etant donné les trois ensembles a, b et c, il existe une correspondance naturelle immédiate entre les deux ensembles fonctionnels $(a \times b) \to c$ et $a \to (b \to c)$. L'addition est classiquement considérée comme un élément de l'ensemble
$(\mathbf{N} \times \mathbf{N}) \to \mathbf{N}$, mais la définition de add à partir de $add(m)$ suggère plutôt qu'il s'agit d'un élément de $\mathbf{N} \to (\mathbf{N} \to \mathbf{N})$: la fonction add associe alors à tout naturel m la fonction $add(m)$, qui elle-même associe à tout naturel n le naturel $add(m)(n) = add(m,n) = m + n$.

La *soustraction naturelle* peut se définir par le biais de fonctions auxiliaires sub_m; on a $sub_m(n) = 0$ si $n \leq m$ et $sub_m(n) = sub1(sub_m(sub1(n)))$ si $n \geq m$; on pose alors

$$sub(n,m) = n \dot{-} m \;=_{def}\; sub_m(n)\,, \text{ pour tous } m,n \in \mathbf{N}.$$

(On écrit parfois $n \dot{-} m$ au lieu de $n - m$ pour éviter la confusion avec la soustraction entière habituelle; on a $5 \dot{-} 7 = 0$ et $5 - 7 = -2$.)

La multiplication se définit par itération. On peut par exemple poser $mult(m,n) = add(m)^{(n)}(0) = (add1^{(m)})^{(n)}(0)$.

La technique de définition récursive n'est pas restreinte aux fonctions de domaine $\mathbf{N}$; nous verrons plus loin que toute fonction dont le domaine est un ensemble bien fondé est susceptible d'être définie récursivement.

2.3.6. *Entiers relatifs*

Une première généralisation de la notion d'entier naturel est celle d'entier relatif. Elle est classiquement introduite de manière syntaxique. La règle de formation est

0 est un entier relatif;
si n est un entier naturel non nul, alors n et $-n$ sont des entiers relatifs.

Cette règle est tacitement complétée par l'affirmation que tout nombre relatif s'obtient d'une et une seule manière au départ de la règle.

Il est alors facile de définir les opérations habituelles, et de montrer les liens existant entre ces opérations et les opérations correspondantes sur les naturels.

Cette manière de procéder ne nous convient pas pour deux raisons, l'une technique et l'autre philosophique. Tout d'abord, la règle de formation implique l'introduction d'un nouveau symbole, $-$, dont le statut n'est pas très clair. Ensuite, ce mode de définition des entiers relatifs, quoique simple, est tout à fait arbitraire et ne traduit pas l'existence d'une motivation particulière pour introduire cette définition. On va donc donner ici une définition plus "naturelle" des entiers relatifs.

L'idée intuitive d'entier négatif est parfois présentée au moyen de la notion de dette. Considérons des couples de naturels, tels (6,3), (2,1) et (4,5). Si l'origine des couples est interprétée comme un avoir initial et l'extrémité comme une dépense ultérieure, chaque couple définit clairement une situation résultante, qui peut être "positive" (reste, crédit) ou "négative" (dette, débit). A chaque couple de naturels (notion bien définie) correspond un entier relatif (notion à définir). On ne peut pas simplement identifier les entiers relatifs aux couples de naturels, parce que, clairement, des couples distincts comme (6,3) et (7,4) doivent définir un même entier relatif.

On définit d'abord une relation R qui mettra en correspondance des couples tels que (6,3) et (7,4) en imposant

$$((x,y),(u,v)) \in R \equiv (x,y,u,v \in \mathbf{N} \wedge x+v = y+u)\,.$$

Cette relation est une relation d'équivalence de domaine $\mathbf{N} \times \mathbf{N}$, et on peut définir l'ensemble $\mathbf{Z}$ des entiers relatifs par

$$\mathbf{Z} =_{def} (\mathbf{N} \times \mathbf{N})/R\,.$$

Il faut alors définir les opérations usuelles sur $\mathbf{Z}$. Nous examinerons surtout ici l'addition, qui est d'abord définie sur $\mathbf{N} \times \mathbf{N}$ comme suit :

$$add_2((x,y),(u,v)) =_{def} (add(x,u), add(y,v))\,.$$

On peut démontrer que la somme de deux couples appartenant à deux classes d'équivalence données z_1 et z_2 appartient toujours à une même troisième classe z_3. Ce fait exprime la compatibilité de l'addition avec la relation d'équivalence R et permet de définir la somme de deux entiers relatifs z_1 et z_2 comme l'entier relatif z_3 constitué des sommes de deux éléments quelconques de z_1 et de z_2.

Les entiers relatifs peuvent être désignés par n'importe lequel de leurs éléments mais on convient généralement de choisir l'unique couple comportant un zéro. Par exemple, l'entier relatif $\{(0,5),(1,6),\ldots\}$ est désigné par $(0,5)$. On pousse souvent l'abus d'écriture un peu plus loin en désignant par n l'entier relatif contenant le couple $(n,0)$ et par $-n$ l'entier relatif contenant le couple $(0,n)$. L'entier relatif contenant $(0,0)$ est noté 0. Ceci revient à assimiler certains entiers relatifs aux entiers naturels, ce que fait aussi la construction classique. Ces abus peuvent paraître graves puisque, par exemple, $(0,0) = \{\{0,0\},\{0\}\} = \{\{0\},\{0\}\} = \{\{0\}\} \neq 0$. En fait, comme nous l'avons déjà signalé, ce n'est pas la nature exacte des nombres qui importe, mais la

façon dont ils sont combinés.[48] De ce point de vue, les abréviations sont licites vu que, si m et n sont des entiers naturels, les couples $add_2((m,0),(n,0))$ et $(add(m,n),0)$ appartiennent au même entier relatif, dénoté $add(m,n)$.[49]

On peut également définir la multiplication des entiers relatifs au moyen de l'opération $mult_2$, sur $\mathbf{N} \times \mathbf{N}$, donnée par

$$mult_2((x,y),(u,v)) =_{def} \\ (add(mult(x,u),mult(y,v)),add(mult(x,v),mult(y,u)))\,,$$

qui traduit simplement l'égalité classique $(x-y)(u-v) = (xu+yv)-(xv+yu)$. On définit alors le produit $mult(z_1,z_2)$ de deux entiers relatifs comme l'entier relatif contenant les produits de deux éléments quelconques de z_1 et de z_2. Comme pour l'addition, cette définition est possible parce que ces produits appartiennent tous à une même classe d'équivalence. Classiquement, le produit de deux entiers z_1 et z_2 est noté $z_1.z_2$ ou $z_1 z_2$.

2.3.7. *Nombres rationnels, réels et complexes*

Les nombres rationnels s'obtiennent à partir des entiers relatifs par un mécanisme semblable à celui qui a permis d'obtenir ceux-ci à partir des entiers naturels. Plus précisément, on définit l'ensemble $\mathbf{Q}$ des rationnels comme le quotient du produit $\mathbf{Z} \times \mathbf{Z}_0$ par la relation d'équivalence définie par :[50]

$$((x,y),(u,v)) \in R \equiv (x,u \in \mathbf{Z} \wedge y,v \in \mathbf{Z}_0 \wedge x.v = y.u)\,.$$

La classe d'équivalence comportant l'élément (a,b) est en fait l'ensemble des fractions égales à celle dont le numérateur est a et le dénominateur est b. On représente habituellement une telle classe par l'unique *fraction réduite* qu'elle contient. La fraction (a,b) est réduite si $b > 0$ et si le seul diviseur positif commun à a et b est 1. On vérifie facilement que chaque classe d'équivalence

[48] Le statut quelque peu secondaire accordé à la *nature* des objets étudiés distingue les mathématiques des autres sciences; le mathématicien étudie les lois qui régissent les objets et leurs combinaisons plutôt que les objets eux-mêmes. Cette attitude peut paraître paradoxale, dans la mesure où la science mathématique est souvent qualifiée de "fondamentale". On peut en inférer que la philosophie est plus fondamentale que les mathématiques, mais on peut aussi considérer que la nature même des nombres s'identifie aux lois de l'arithmétique; ce débat est du ressort de la philosophie des sciences. D'un point de vue plus pragmatique, on observe que cette éventuelle carence des mathématiques, loin de nuire à leur intérêt intrinsèque ou à leur utilité, semble parfois la condition *sine qua non* de cet intérêt et de cette utilité. Cette opinion est bien illustrée par la théorie des ensembles; la structure d'un ensemble a souvent plus d'importance que la nature de ses éléments. (Nous avons déjà cité quelques exemples importants : les objets géométriques, les ensembles numériques et le produit cartésien.) Dans le cadre axiomatique abordé plus loin, *l'axiome de renommage* ou *de remplacement* confortera et précisera ce point.

[49] Le phénomène d'abréviation est omniprésent en mathématique; il convient certes d'être prudent, mais interdire les abréviations est inacceptable en pratique.

[50] $\mathbf{Z}_0$ désigne l'ensemble des entiers relatifs à l'exclusion de zéro; on introduit donc l'abréviation $\mathbf{Z}_0 =_{def} \mathbf{Z} \setminus \{0\}$.

comporte une et une seule fraction réduite. L'ensemble $\mathbf{Q}$ est donc (assimilable à) l'ensemble des fractions réduites. L'ensemble $\mathbf{Z}$ peut être considéré comme le sous-ensemble de $\mathbf{Q}$ comportant (les classes d'équivalence représentées par) les fractions dont le dénominateur vaut 1; une telle fraction est toujours réduite. On définit aisément l'addition et la multiplication dans l'ensemble $\mathbf{Q}$.

La définition de l'ensemble des nombres réels à partir de celui des rationnels est plus délicate mais, à nouveau, l'approche naïve est un bon guide. Cette application de la théorie des ensembles à la modélisation d'un objet mathématique difficile illustre bien le principal mérite de la théorie, qui est de préciser les idées intuitives des mathématiciens. Nous passerons en revue trois approches, dues respectivement à Cantor, Dedekind et Weierstrass.

L'approche de Cantor se base sur l'idée intuitive que, quelle que soit la nature exacte d'un nombre réel, il doit être "approximable" par des rationnels. Il faut d'abord formaliser le concept d'approximation; pour ce faire, nous considérons d'abord un exemple. Le rationnel $1/3$ admet le développement décimal $0,33...3...$ (ce développement est illimité et périodique). Une approximation "au centième" est donnée par la formule $1/3 \in [0.33, 0.34]$. On voit (et on démontre aisément) que le rationnel $1/3$ est l'unique élément appartenant à l'intersection de la suite $(I_n : n \in \mathbf{N})$ d'intervalles *emboîtés* ($I_{n+1} \subset I_n$, pour tout n) telle que I_n est $[0.33...3, 0.33...4]$, l'origine et l'extrémité de l'intervalle contenant chacun n décimales.

On sait que des "nombres" tels que $\sqrt{2}$, π et e, pour lesquels nous disposons de représentations géométriques simples, ne sont pas des rationnels; en effet, leur développement décimal n'est pas périodique, alors que la périodicité caractérise les rationnels. Ceci conduit à la définition de Cantor: un *nombre réel* est (défini par) une suite d'intervalles fermés[51] emboîtés dont le diamètre tend vers 0. La notion "tendre vers 0" est aisément formalisée au moyen de la formulation classique en ε et δ. Deux suites d'intervalles $([a_n, b_n])$ et $([c_n, d_n])$ définissent le même nombre réel si les suites de nombres rationnels $(a_n - c_n)$ et $(b_n - d_n)$ tendent toutes deux vers 0. La définition de Cantor est intuitive; elle formalise la notion de nombre décimal illimité, sans être dépendante de la numération décimale.[52] Cette notion elle-même provient de l'intuition de nombres issus de la géométrie (ou de l'analyse) tels que $\sqrt{2}, \pi$ et e, dont les mathématiciens ont assez tôt suspecté, puis démontré, le caractère non rationnel.

L'approche de Dedekind repose sur le fait qu'à tout rationnel $r \in \mathbf{Q}$ correspondent deux *coupures*, ou *segments initiaux*, qui le caractérisent. Dans un ensemble totalement ordonné, une coupure ou segment initial C est un

[51] Un intervalle fermé (de rationnels) est un ensemble du type $[a, b] = \{x \in \mathbf{Q} : a \leq x \leq b\}$. Le diamètre de $[a, b]$ est 0 si $a \geq b$ et $b - a$ sinon.

[52] Le développement décimal correspond au cas particulier où le diamètre de I_n est 10^{-n}, et où les bornes sont des décimaux limités d'ordre n.

ensemble qui contient tous les minorants de ses éléments: si $a \in C$ et $b < a$, alors $b \in C$. Les coupures caractérisant le rationnel r sont $\{s : s \in \mathbf{Q} \wedge s < r\}$ et $\{s : s \in \mathbf{Q} \wedge s \leq r\}$. On observe que, si tout rationnel définit une, ou même deux coupures, certaines coupures, telles $\{x : x < 0 \vee x^2 < 2\}$ et $\{x : x < 0 \vee x^2 \leq 2\}$, ne définissent pas de rationnels. L'idée de Dedekind est de décréter que ces coupures "définissent" un nombre réel (ici $\sqrt{2}$). Plus formellement, un nombre réel *est* une coupure. L'interprétation géométrique justifie le terme "coupure". Si les nombres rationnels sont représentés sur une droite, une coupure est simplement une partition en deux demi-droites.

L'approche de Weierstrass, comme celles de Cantor et de Dedekind, vise à "compléter", à "combler les lacunes" de l'ensemble des rationnels. Considérons une suite de rationnel (r_n), convergeant vers le rationnel r. On sait qu'une suite convergente possède la propriété de Cauchy, c'est-à-dire que $|r_p - r_q|$ tend vers 0 quand $min(p, q)$ tend vers l'infini. A nouveau, dans l'ensemble $\mathbf{Q}$, la réciproque n'est pas vraie: il existe des suites de Cauchy de rationnels qui ne convergent pas vers un rationnel (un exemple est la suite $(3, 3.14, 3.141, \ldots)$ des développements décimaux de π). On peut "compléter" l'ensemble des rationnels en décrétant que toute suite de Cauchy de rationnels "définit" (s'identifie à) un nombre réel. Plusieurs suite de Cauchy correspondront au même nombre réel, comme s'était déjà le cas pour les suites d'intervalles emboîtés. On pose donc $\mathbf{R} =_{def} \mathcal{C}(\mathbf{Q})/\rho$, où $\mathbf{R}$ est l'ensemble des *nombres réels*, $\mathcal{C}(\mathbf{Q})$ est l'ensemble des suites de Cauchy de rationnels et ρ est la relation d'équivalence sur $\mathcal{C}(\mathbf{Q})$ définie par $((a_n), (b_n)) \in \rho$ si et seulement si $|a_n - b_n|$ tend vers 0 quand n tend vers l'infini.

On peut prouver, mais nous ne le ferons pas ici, que ces trois définitions sont équivalentes. Il faut aussi montrer qu'elles sont opérationnelles, c'est-à-dire qu'elles permettent de définir les opérations classiques appliquées aux nombres réels. Par exemple, on définit l'addition des réels à partir de celle des rationnels; si les suites de Cauchy (a_n) et (b_n) définissent respectivement les réels a et b, alors la suite $(a_n + b_n)$ est également de Cauchy et définit le réel $(a + b)$. On démontre aisément la consistance de cette définition. De même, si C et D sont des coupures, la coupure $C + D$ sera $\{x + y : x \in C \wedge y \in D\}$. Enfin, si les suites d'intervalles emboîtés $([a_n, b_n])$ et $([c_n, d_n])$ définissent des nombres réels, la suite $([a_n + c_n, b_n + d_n])$ définit également un nombre réel, qui est la somme. Les autres opérations sur les réels s'introduisent d'une manière similaire (attention cependant que si la suite (c_n) est convergente, la suite $(1/c_n)$ ne l'est que si $\lim c_n \neq 0$; ceci traduit le fait que 0 n'a d'inverse ni dans $\mathbf{Q}$ ni dans $\mathbf{R}$).

Dans notre contexte, l'important n'est pas de connaître et de justifier ces définitions, mais plutôt d'observer que la théorie des ensembles précise l'intuition et s'accorde avec elle (contrairement à une opinion répandue). Dans le cas des nombres réels, les trois définitions que nous avons formalisées sont inspirées par la volonté de "combler les lacunes" des rationnels. L'approche

ensembliste est extrêmement simple; on décrit les lacunes jugées gênantes, puis on décrète que l'objet mathématique caractérisant une lacune est un nombre réel. Dans l'approche de Cantor, cet objet est l'intersection d'une suite d'intervalle emboîtés de diamètre tendant vers 0; pour Dedekind, une lacune est (définie par) une coupure des rationnels, c'est-à-dire par une bipartition des rationnels telle que tous les éléments d'une composante sont plus petits que tous les éléments de l'autre composante. Enfin, l'approche de Cauchy-Weierstrass consiste à décréter que chaque suite de Cauchy de rationnels doit avoir une limite (rationnelle si la suite converge dans $\mathbf{Q}$, irrationnelle sinon). Quelle que soit l'approche utilisée, les nombres réels *complètent* les rationnels. Cette aptitude du langage ensembliste à formaliser toute construction est à la base de l'utilité particulière de la théorie des ensembles. L'emploi d'un langage commun favorise les comparaisons et on peut, en particulier, prouver que les trois approches que nous avons évoquées définissent des ensembles isomorphes, ayant le même comportement vis-à-vis des opérations arithmétiques et de la relation d'ordre numérique.

Des méthodes ensemblistes analogues permettent de définir les nombres complexes. On peut aussi définir les points (couples ou n-uplets de nombres réels) et les autres objets géométriques.

2.4. Approche formelle

Jusqu'ici, la théorie des ensembles paraît fort simple. A partir de deux schémas d'axiomes (extensionnalité et compréhension), on a obtenu une théorie assez expressive pour décrire le comportement des objets mathématiques usuels. Un résultat aussi considérable, obtenu aussi économiquement, justifie le nom de "paradis" parfois attribué à la théorie des ensembles, telle qu'elle a été développée au siècle dernier, par Cantor principalement.

Malheureusement, il est apparu fort tôt que la théorie des ensembles, sous sa forme naïve, n'est pas satisfaisante; plus précisément, nous allons voir qu'elle permet la démonstration d'énoncés contradictoires ... ce qui est inacceptable!

2.4.1. *Paradoxes*

L'une des premières contradictions obtenues, et aussi l'une des plus simples, est une traduction du vieux paradoxe du barbier.[53] Le point de départ est une instance du schéma de compréhension, obtenu en prenant pour $P(x)$ la formule $x \notin x$. On obtient le théorème

$$\exists! y \, \forall x \, [x \in y \equiv x \notin x] \,,$$

dont on déduit successivement

[53] "Le barbier rase tous les hommes qui ne se rasent pas eux-mêmes. Se rase-t-il lui-même?"

$$\exists y \forall x [x \in y \equiv x \notin x],$$
$$\exists y [y \in y \equiv y \notin y],$$
$$\mathit{false}.$$

On retrouve le paradoxe du barbier en interprétant le symbole $\in$ comme la relation "est rasé par"; l'énoncé paradoxal affirme l'existence d'un barbier y. Le principe de compréhension conduit donc à une contradiction. Ce phénomène est le *paradoxe de Russell.* Il montre que la théorie esquissée est inconsistante. Plus exactement, il est abusif de dire que toute propriété $P(x)$ détermine un ensemble $\{x : P(x)\}$. Les paradoxes de la théorie naïve des ensembles (d'autres exemples seront évoqués plus loin) ont conduit au développement d'une théorie axiomatique que nous présentons dans la suite de ce chapitre.

La théorie axiomatique devra isoler, parmi les collections, celles qui ne provoquent pas d'inconsistance. Seules ces collections particulières recevront le nom d'*ensemble.* On ne connaît malheureusement pas de moyen de détecter a priori quelles propriétés sont susceptibles de donner lieu à des collections paradoxales; pour contourner cette difficulté, on étendra la portée du point de vue constructif déjà évoqué. Jusqu'ici, nous avons adopté un type primitif "ensemble", et nous l'avons utilisé pour définir, de façon constructive, d'autres types mathématiques (relations et nombres). Désormais, nous ne considérons plus les ensembles comme primitifs. Tous les ensembles devront être construits par application de certaines règles à des ensembles déjà obtenus.[54]

La détermination des règles de construction admissibles est délicate mais la théorie naïve est un bon guide: les opérations classiques de formation d'ensembles (réunion, produit cartésien, etc.) doivent être admises. Sans cela, la théorie obtenue serait trop pauvre pour être utile, et s'écarterait trop de la notion intuitive d'ensemble.

On appellera *collection* tout objet considéré comme une multiplicité de composants, que cette multiplicité puisse ou non conduire à des paradoxes. A ce titre, l'objet $C = \{x : x \notin x\}$ introduit plus haut est une collection (paradoxale). On appelle *classe* une collection dont les éléments sont des ensembles. On a vu plus haut qu'il est souhaitable d'éliminer les éléments fondamentaux en les modélisant par des ensembles appropriés. Ce point de vue sera conservé, ce qui implique que tout ensemble est une classe. La réciproque est fausse; l'objet C, appelé *classe de Russell*, est un contre-exemple.

Remarque. On ne perd rien à imposer la qualité d'ensemble à tout élément. En effet, on a vu plus haut que les nombres pouvaient être modélisés par des ensembles (il s'agissait d'ensembles "naïfs" mais on veillera à ce que ces objets demeurent des ensembles dans l'approche axiomatique). Comme, d'autre part, on conçoit difficilement des objets qui ne puissent être étiquetés et désignés

[54] Comme nous le verrons, seule la collection "incontournable" des entiers naturels recevra directement le statut d'ensemble. Tout autre ensemble sera nécessairement construit à partir de $\mathbf{N}$, au moyen de règles rigides et peu nombreuses.

par des nombres, ou des familles de nombres, on ne voit pas pourquoi on devrait étudier autre chose que les ensembles obtenus à partir des nombres naturels obtenus plus haut. Ceci n'est pas une justification rigoureuse, mais simplement une explication, d'une limitation un peu bizarre à première vue. Signalons d'ailleurs que les éléments, comme les collections, sont susceptibles de provoquer des paradoxes; c'est une bonne raison pour éviter l'introduction d'éléments arbitraires.

Les paradoxes recensés dans la théorie naïve sont généralement liés à la définition, via le schéma de compréhension, de collections infinies. Ceci suggère qu'il convient de se méfier de ce genre de collection. Il n'est toutefois pas possible de les exclure, ce qui empêcherait notamment de définir les ensembles numériques. Simplement, on donnera a priori le statut d'ensemble à une seule collection infinie, et tout ensemble infini devra être construit à partir de cette collection-là. Les moyens de construction autorisés ne pourront attribuer la qualité d'ensemble infini à un objet que si ces moyens sont appliqués à des ensembles infinis déjà obtenus.

L'approche habituelle consiste à ne considérer que les ensembles, en excluant les classes plus générales. Le concept de classe n'est pourtant pas, dans l'état actuel des connaissances, paradoxal en lui-même: les difficultés n'apparaissent que lorsqu'une classe est en position d'élément d'une collection. Nous nous contenterons donc d'éliminer ce cas. Cela suffit à résoudre le paradoxe de Russell: la classe C n'est pas un ensemble et n'appartient donc à aucune classe. On a donc $\forall x\,(C \notin x)$ et, en particulier, $C \notin C$.

Il peut paraître curieux, d'un point de vue philosophique, de se méfier systématiquement de l'infini. On peut toutefois justifier cette attitude par l'abondance des paradoxes que les mathématiciens ont découverts au cours du temps. En particulier, le vieux principe "le tout est plus grand que les parties" ne s'applique pas immédiatement aux quantités infinies. Par exemple, tout naturel possède un carré, alors que tout naturel n'est pas un carré; il existe néanmoins une bijection entre la classe $\mathbf{N}$ et sa sous-classe $\{x : x \in \mathbf{N} \wedge \exists y(y \in \mathbf{N} \wedge y = x^2)\}$. Cette particularité est caractéristique des classes infinies. Un autre exemple est celui des séries. Il est bien connu que l'addition est une opération commutative et associative ... mais cela ne s'étend pas sans précautions aux sommes infinies. Considérons par exemple la série

$$1 - 1/2 + 1/3 - 1/4 + 1/5 - \cdots$$

On peut, en permutant et regroupant les termes, lui attribuer n'importe quelle valeur réelle ou la rendre divergente.

2.4.2. *L'approche constructive abstraite en informatique*

Nous avons vu au paragraphe 2.3.1 comment le type "pile" pouvait être représenté par le type plus concret "tableau", ce qui permet de manipuler des

piles dans les langages de programmation classiques comme le Fortran. Nous avons de même représenté le type "nombre naturel" par le type "ensemble", en observant notamment que $n+1$ devenait $n \cup \{n\}$ et que $n-1$ devenait $\bigcup n$.

Dans le cas de la pile, nous avons d'abord énoncé les propriétés caractéristiques de la pile, qui ont guidé le processus de représentation proprement dit. Certains langages de programmation modernes (ML notamment) permettent de n'effectuer que la première étape. Dès que l'utilisateur-programmeur d'un système ML a fourni les axiomes de la pile, il peut effectivement manipuler des piles, sans ce soucier de savoir si, quelque part, ces piles sont représentées par des tableaux ou autrement.

Cette approche abstraite de la définition des types de données est simple et élégante. Elle n'exige pas (en principe) la préexistence d'un type "primitif". Les premiers axiomes de Peano (§ 2.3.3) peuvent être vus comme une définition d'un type abstrait "nombre naturel".

L'approche concrète nous a servi à représenter tous les types mathématiques par le type universel "ensemble", mais l'approche abstraite, par axiomes, nous sera nécessaire pour définir le type ensemble lui-même.

L'approche abstraite consiste donc à appréhender (à définir ?) des objets au moyen de propriétés axiomatiques. Plus précisément, les objets sont spécifiés indirectement, au moyen des opérations par lesquelles ils peuvent être combinés.

2.4.3. *Vers une théorie formelle*

On peut à présent définir formellement la théorie des ensembles.

La syntaxe de la théorie (axiomatique) des ensembles est très simple. La signature comporte une constante individuelle, V, appelée *classe universelle*, et un prédicat binaire, $\in$, appelé prédicat d'appartenance. Ceci donne lieu aux règles de formation primitives suivantes :

- V est une constante.
- La constante et les variables sont des termes.
- Si u et v sont des termes, alors $u \in v$ et $u = v$ sont des formules atomiques.

Conformément à l'usage, on utilise la notation infixée pour le prédicat d'appartenance.

Le premier schéma d'axiome est le schéma d'*extensionnalité*, déjà introduit :

(2.2) $\quad \forall x \forall y \, [x = y \;\equiv\; \forall z \, (z \in x \;\equiv\; z \in y)] \, .$

Le second est celui de *séparation* :

(2.3) $\quad \exists! y \forall x \, [x \in y \;\equiv\; (x \in V \;\wedge\; P(x))] \, .$

Rappelons que $\exists! x \, Q(x)$ est une écriture abrégée de la formule

$$\exists x \, [Q(x) \;\wedge\; \forall y \, (Q(y) \Rightarrow y = x)] \, .$$

Ce schéma est directement inspiré du schéma de compréhension introduit (et rejeté) précédemment. La notation $P(x)$ désigne toute formule admettant x

comme seule variable libre, et ne comportant pas d'occurrence de y. Par rapport au schéma de compréhension, on observe l'introduction du "garde-fou" $x \in V$; plus précisément, l'écriture $x \in y$ ne sera employée que si x est un élément d'une (vaste) classe, notée V. Intuitivement, V représente la classe de tous les ensembles et l'axiome affirme l'existence et l'unicité d'une sous-classe constituée des ensembles x vérifiant $P(x)$. En fait, la classe V est "séparée" en deux sous-classes, selon le critère $P(x)$.

Toute instance du schéma de séparation affirme l'existence et l'unicité d'un objet de la théorie, c'est-à-dire d'une classe. Si on se donne le droit d'introduire un terme pour désigner cette classe, on choisira naturellement la notation $\{x : P(x) \wedge x \in V\}$. Il est commode d'étendre cette notation et de permettre l'écriture $\{x : P(x)\}$ même si $P(x)$ n'implique pas que x soit un ensemble, c'est-à-dire un élément de V. Ceci peut se faire, à condition de donner un sens précis à cette notation. Le *schéma d'axiome de Church* est introduit dans ce but :

(2.4) $\quad \forall y\,[y \in \{x : P(x)\} \equiv (y \in V \wedge P(y))]\,.$

Si on considère la notation $\{x : P(x)\}$ comme primitive, le schéma d'axiome de Church remplace le schéma de séparation; sinon, on voit immédiatement que le schéma de Church permet la règle de formation dérivée :

- Si $P(x)$ est une formule dont x est la seule variable libre, alors $\{x : P(x)\}$ est un terme.

On peut maintenant définir la classe de Russell $C =_{def} \{x : x \notin x\}$. L'instance correspondante du schéma de Church est

$$\forall y\,[y \in C \equiv (y \in V \wedge y \notin y)]\,.$$

On déduit de cet énoncé le théorème

$$C \in C \equiv (C \in V \wedge C \notin C)\,,$$

qui se simplifie immédiatement en le théorème

$$C \notin V \wedge C \notin C\,.$$

On dira que C est une *classe propre*, c'est-à-dire une classe qui n'est pas un ensemble.

On peut introduire les notations suivantes :

$$ens(x) =_{def} x \in V\,,$$
$$clp(x) =_{def} x \notin V\,.$$

Les éléments d'une classe sont toujours des ensembles; cela résulte du schéma de Church. Voici une conséquence importante de ce fait. Si $\{a\}$ désigne la classe $\{x : x = a\}$, on a

$$\forall x\,[x \in \{a\} \equiv (ens(x) \wedge x = a)]\,;$$

si a est une classe propre, l'énoncé se réduit à

$$\forall x\,[x \notin \{a\}]\,,$$

ce qui signifie que $\{a\}$ est vide. (Si par contre a est un ensemble, a est l'unique élément de l'ensemble $\{a\}$.)

Le fait que la notation $\{x : P(x)\}$ désigne une classe et non une quelconque collection peut s'écrire :

$$\{x : P(x)\} = \{x : (P(x) \wedge x \in V)\}.$$

Les objets de l'univers, c'est-à-dire du domaine d'interprétation de la théorie, sont les classes (ensembles et classes propres). L'ensemble vide sera le seul élément fondamental, et le principe d'extensionnalité est respecté sans restriction : deux classes sont égales si et seulement si elles contiennent les mêmes éléments. Ceci permet a posteriori de considérer le symbole d'égalité (classiquement considéré comme primitif dans une théorie du premier ordre) comme un simple mécanisme abréviatif : l'écriture $a = b$ représente la formule $\forall x\,(x \in a \equiv x \in b)$. Observons aussi que les abréviations introduites au paragraphe 2.2.2 ne posent aucun problème et restent valables.

Remarque. La plupart des auteurs, pour des raisons esthétiques et/ou pédagogiques, basent la théorie axiomatique des ensembles sur le seul symbole d'appartenance. Ils remplacent alors l'écriture $x \in V$ par l'écriture $\exists y\,(x \in y)$. Nous avons préféré faciliter au maximum l'évocation explicite des classes propres. D'abord, l'expérience montre qu'il n'est pas possible ni même souhaitable d'éviter complètement cette notion assez délicate. Bien plus, c'est l'existence même des classes propres qui a motivé la formalisation de la notion d'ensemble. Enfin, certaines classes propres jouent un rôle important dans la théorie, à commencer par la classe des ordinaux (§ 2.5) et les fonctions dont le domaine est la classe des ordinaux. Il ne faudrait cependant pas croire que la notion de classe généralise radicalement celle d'ensemble. Techniquement, le gain est faible; en particulier, toutes les collections ne sont pas des classes. On peut par exemple regrouper (informellement) en une collection les classes propres déjà évoquées. Cette collection contiendrait (au sens habituel, informel du terme) notamment la classe universelle, la classe des triplets, celle des ensembles inductifs, la classe de Russell, la classe des ordinaux et celle des cardinaux. Une telle collection n'est pas une classe, puisqu'une classe ne contient (au sens formel comme au sens informel) que des ensembles.

2.4.4. *Construction axiomatique des ensembles*

Les schémas intuitifs d'extension et de compréhension constituent la base de la théorie naïve des ensembles. Le schéma d'extension, le schéma de séparation et le schéma de Church constituent la base de la théorie axiomatique. Il faut cependant remarquer que cette base ne permet pas de construire effectivement des ensembles. Plus précisément, nous ne disposons pas de termes (en dehors du terme primitif V) dénotant des ensembles particuliers. Des axiomes supplémentaires permettront de définir des ensembles particuliers, qui s'écriront

$\{x : P(x)\}$ pour des formules $P(x)$ adéquates. Avant de s'engager dans cette voie, rappelons que, si l'existence et l'unicité d'un objet sont établies, il est licite d'introduire une notation particulière pour désigner cet objet; en effet, cette notation pourra toujours être éliminée. On peut aller un peu plus loin, et s'autoriser à utiliser n'importe quelle notation, par exemple $\{x : P(x)\}$, pour peu qu'une signification précise lui soit attribuée. Le schéma de Church donne une signification à la formule $y \in \{x : P(x)\}$ et, vu le principe d'extensionnalité, il suffit que cette formule ait un sens pour que le terme $\{x : P(x)\}$ en ait un aussi.

Cette justification sémantique est suffisante pour développer la théorie des ensembles pour elle-même. Par contre, elle n'est pas suffisante si on souhaite attribuer à la théorie des ensembles le statut de théorie du premier ordre. Dans une telle théorie, en effet, les termes sont construits exclusivement à l'aide de termes primitifs (constantes individuelles) et d'applications à ceux-ci ou aux variables de foncteurs particuliers (les constantes fonctionnelles).

De ce point de vue, la seule classe dénotable est la classe universelle V. Il ne servirait à rien d'introduire d'autres constantes individuelles, et introduire des constantes fonctionnelles rendrait pléthorique le langage de la théorie, et inutilisable la théorie elle-même.

Un bon compromis est d'accepter l'usage de la notation $\{x : P(x)\}$ tout en observant que, en principe, elle n'est pas indispensable. En fait, on peut développer la théorie des ensembles sans utiliser de notation explicite pour les désigner. Par exemple, une formule du type $a \in \{x : P(x)\}$ peut se récrire $ens(a) \wedge P(a)$. De même, la formule $\{x : P(x)\} \in a$ peut se récrire

$$\exists y\,[y \in a \wedge \forall x\,(x \in y \equiv [ens(x) \wedge P(x)])]\,.$$

(Rappelons que $ens(x)$ peut être remplacé par $x \in V$.) Ce procédé est systématique et, le symbole d'appartenance étant l'unique symbole extralogique indispensable de la théorie,[55] on peut prouver par induction sur la structure des formules que toute formule comportant un terme faisant intervenir la notation de classe peut se récrire sans cette notation. Bien évidemment, cette élimination est fastidieuse et nuisible en pratique; il est néanmoins important qu'elle puisse se faire en théorie.

2.4.5. *Construction des ensembles finis*

On va utiliser la technique des types abstraits de données (§ 2.4.2) pour définir axiomatiquement les ensembles. Les axiomes seront conçus sur les modèles suivants :

- Le terme t est un ensemble.

[55] la constante V elle-même n'est pas indispensable; on peut toujours remplacer $x \in V$ par $\exists y\,(x \in y)$.

- Si $t_1, \ldots, t_n$ sont des ensembles, alors $F(t_1, \ldots, t_n)$ est un ensemble. (F désigne ici un combinateur à n places.)

Formellement, on écrira

- $ens(t)$.
- $\forall x_1 \ldots \forall x_n [(ens(x_1) \wedge \cdots \wedge ens(x_n)) \Rightarrow ens(F(x_1, \ldots, x_n))]$.

Une première famille d'axiomes potentiels est la suivante:

- $\emptyset$ est un ensemble.
- Si a et b sont des ensembles,
  alors $\{a, b\}$, (a, b), $a \cup b$, $a \cap b$, $a \setminus b$, $a \times b$, $\bigcup a$, $\bigcap a$, $\mathcal{P}(a)$ sont des ensembles.

Comme on l'a mentionné plus haut, on s'est laissé guider par la théorie naïve pour décréter que les opérateurs usuels respectent la qualité d'ensemble. Cette approche n'est pourtant pas idéale. Les opérateurs de la théorie naïve sont nombreux, ce qui favorise les faux pas; par exemple, la classe $\bigcap \emptyset$, quoiqu'obtenue à partir d'un ensemble et au moyen d'un opérateur "simple", n'est pas un ensemble (puisque c'est la classe universelle). De plus, il est souhaitable de permettre explicitement l'opération de "renommage" ou de "remplacement": si certains éléments de l'ensemble a sont remplacés par d'autres éléments, la classe a' ainsi obtenue restera un ensemble. L'idée intuitive sous-jacente est que seule une classe de taille excessive (le concept de taille sera formalisé plus loin) peut provoquer l'apparition de paradoxes et être une classe propre. Remplacer un ensemble x, élément de a, par un autre ensemble $f(x)$ dans l'ensemble a ne peut augmenter la taille de a (il peut la diminuer si $f(x) \in a \setminus \{x\}$).

Pour éliminer la première objection, on s'orientera vers une famille minimale d'axiomes constructifs, énumérés ci-dessous.

$$
\begin{array}{ll}
ens(\emptyset) & \text{(vide)}, \\
\forall x \forall y [(ens(x) \wedge ens(y)) \Rightarrow ens(\{x, y\})] & \text{(paire)}, \\
\forall x [(ens(x) \Rightarrow ens(\bigcup(x))] & \text{(réunion)}, \\
\forall x [(ens(x) \Rightarrow ens(\mathcal{P}(x))] & \text{(puissance)}.
\end{array} \tag{2.5}
$$

Ces axiomes permettent de construire des ensembles de plus en plus grands. Observons qu'une paire (ou, plus généralement, un n-uplet non ordonné) est toujours un ensemble, même si ses composants ne le sont pas. En effet, si a est une classe propre, alors $\{a\}$ est l'ensemble vide. Plus généralement, si $a_1, \ldots, a_n$ sont des classes propres, alors $\{a_1, \ldots, a_n, b_1, \ldots\}$ se réduit à $\{b_1, \ldots\}$. Rappelons aussi que (a, b) est une abréviation de $\{\{a, b\}, \{a\}\}$; un couple est donc toujours un ensemble.

La faculté de renommage ou de remplacement est formalisée par l'axiome suivant:

$$\forall f [(fonc(f) \wedge ens(dom(f))) \Rightarrow ens(cod(f))] \qquad \text{(remplacement)}. \tag{2.6}$$

Si le domaine d'une fonction est un ensemble, alors le codomaine est également un ensemble. Pour être précis, f n'est une fonction de renommage que si f

est une bijection : le principe même d'une nomenclature précise est que des objets distincts portent des noms distincts. Si on n'impose pas à f d'être une bijection, le codomaine de f peut être plus réduit que le domaine; l'axiome de remplacement justifie donc la réduction de taille et on prouvera plus loin que toute sous-classe d'un ensemble est un ensemble. L'axiome ne justifie pas les extensions puisque f est une fonction et non une relation quelconque.

Remarque. Dans le cadre de la théorie axiomatique, une relation est une classe de couples d'*ensembles*; on a donc

$$\forall x\,[rel(x) \equiv \forall y\,(y \in x \Rightarrow \exists u \exists v\,[ens(u) \wedge ens(v) \wedge y = (u,v)])]\,,$$
$$\forall x\,[fonc(x) \equiv (rel(x) \wedge \forall u \forall y \forall z\,[(ens(y) \wedge ens(z) \wedge (u,y) \in x \wedge (u,z) \in x) \Rightarrow y = z])]\,.$$

Remarques. Intuitivement, le renommage et le remplacement sont des notions différentes. En cas de renommage, l'objet subsiste mais son nom change; en cas de remplacement, l'objet change. Dans le cadre de la théorie des ensembles, la distinction est moins nette. Considérons le cas des deux ensembles $\prod\{a,b\}$ et $a \times b$; ces ensembles sont le domaine et le codomaine d'une même bijection f (§ 2.2.4) tout en étant distincts. Formellement, le second ensemble est obtenu en *remplaçant* dans le premier chaque élément x par son correspondant $f(x)$. Pourtant, les deux ensembles sont appelés "produit cartésien de a et de b", un concept dont on souhaite naturellement l'unicité. Le point de vue algébrique moderne consiste à considérer que le produit cartésien de a et de b est une entité abstraite, dont les ensembles $\prod\{a,b\}$ et $a \times b$ sont des réalisations, ou même simplement des dénominations. Une telle approche est systématique en théorie des catégories; elle se retrouve aussi en informatique. Les "types abstraits" de certains langages de programmation comme ML et Miranda sont analogues au produit cartésien; seules comptent les propriétés d'interaction entre un type abstrait et ses types de base; la manière dont le type est concrétisé est accessoire.

2.4.6. *Quelques théorèmes importants*

Les axiomes donnés plus haut sont dits "constructifs" car ils transmettent la qualité d'ensemble à des classes construites au moyen d'ensembles. Cette approche est suggérée par l'impression intuitive que seules les "très grandes" classes sont susceptibles de donner lieu à des paradoxes, donc de ne pas être des ensembles. Cette impression est en partie validée par le résultat suivant.[56]

Théorème 2.3. Toute sous-classe d'un ensemble est un ensemble.

Nous allons donner de ce théorème une preuve intuitive, puis une preuve semi-formelle. Dans ce contexte, le mot "intuitif" ne s'oppose pas à "rigoureux":

[56] Le terme allemand "Aussonderung" est souvent employé pour désigner cet énoncé.

les arguments sont valides, mais leur expression est informelle. Une preuve semi-formelle est une esquisse de preuve formelle.

Preuve intuitive. Soit b une sous-classe de l'ensemble a. Si b est vide, alors b est un ensemble. Si au contraire b contient un élément c, alors on utilise l'axiome de remplacement pour prouver que b est un ensemble, c'est-à-dire que l'on construit une fonction f de domaine a et de codomaine b. Il suffit de poser $f(x) = x$ si $x \in b$ et $f(x) = c$ si $x \in (a \setminus b)$.

Preuve semi-formelle. Dans le développement qui suit, chaque formule est une conséquence logique des axiomes de la théorie des ensembles, munie des hypothèses $ens(a)$ et $b \subset a$.

1. $b = \emptyset \vee b \neq \emptyset$
2. $b \neq \emptyset \equiv \exists x\,(x \in b)$
3. $b = \emptyset \vee \exists x\,(x \in b)$ (1, 2)
4. $b = \emptyset \Rightarrow ens(b)$
5. $ens(c) \Rightarrow fonc(\{(u,v) : u \in a \wedge (u \in b \Rightarrow v = u) \wedge (u \notin b \Rightarrow v = c)\})$
6. $c \in b \Rightarrow fonc(\{(u,v) : u \in a \wedge (u \in b \Rightarrow v = u) \wedge (u \notin b \Rightarrow v = c)\})$ (5)
7. $c \in b \Rightarrow dom(\{(u,v) : u \in a \wedge (u \in b \Rightarrow v = u) \wedge (u \notin b \Rightarrow v = c)\}) = a$ (6)
8. $c \in b \Rightarrow cod(\{(u,v) : u \in a \wedge (u \in b \Rightarrow v = u) \wedge (u \notin b \Rightarrow v = c)\}) = b$ (6)
9. $c \in b \Rightarrow ens(b)$ (7, 8)
10. $\exists x\,(x \in b) \Rightarrow ens(b)$ (9)
11. $ens(b)$ (3, 4, 10)

Remarques. L'écriture $\{(u,v) : P(u,v)\}$ est une notation abrégée pour le terme $\{x : \exists u \exists v\,[x = (u,v) \wedge P(u,v)]\}$. Une telle abréviation n'est pas dangereuse dans la mesure où un couple n'est jamais une classe propre; il est cependant souhaitable de ne l'utiliser que dans les cas où $P(u,v)$ implique que u et v sont des ensembles. Plus généralement, on admet que l'écriture $\{t(x,y,z) : P(x,y,z)\}$ pour représenter le terme $\{u : \exists x \exists y \exists z\,[u = t(x,y,z) \wedge P(x,y,z)]\}$ si les classes x, y, z et $t(x,y,z)$ sont des ensembles dès que $P(x,y,z)$ est vrai. La règle permettant de prouver la formule $\exists x\,P(x,a) \Rightarrow Q(a)$ à partir de la formule $P(c,a) \Rightarrow Q(a)$ est une règle d'inférence dérivée du calcul des prédicats. Dans cette règle, $P(\alpha,\beta)$ et $Q(\gamma)$ peuvent être non seulement des constantes prédicatives, mais aussi des formules plus générales; elles ne peuvent contenir d'autres variables libres que α et β pour la première formule, et γ pour la seconde. La constante c ne peut pas intervenir dans ces formules, ni dans le terme a. Dans le contexte de la preuve précédente, appliquer cette règle revient simplement à dire "si b n'est pas vide, alors soit c appartenant à b". Cette "règle du choix" d'un élément dans un ensemble non vide ne doit pas être confondue avec "l'axiome du choix" introduit plus loin.

Théorème 2.4. Une fonction est un ensemble si et seulement si son domaine est un ensemble.

Principe de preuve. On applique l'axiome de remplacement à une fonction g dont le domaine est $dom(f)$ et dont le codomaine est f.

Théorème 2.5. Tout produit cartésien est un ensemble.

Principe de preuve. Si E est une classe propre, $\prod E$ est vide, car toute fonction de domaine E est une classe propre, n'appartenant à aucune classe. Si E est un ensemble, tout élément de la classe $\prod E$ est caractérisé par un sous-ensemble de $\bigcup E$ (au moins si les éléments de E sont deux à deux disjoints). Il existe donc une fonction dont le domaine est inclus dans l'ensemble $\mathcal{P}(\bigcup E)$ et dont le codomaine est $\prod E$, ce qui montre que $\prod E$ est un ensemble.
Plus directement, tout élément $f \in \prod E$ se compose de couples de type (x, y), avec $x \in E$ et $y \in \bigcup E$, d'où $(x, y) = \{\{x, y\}, \{x\}\} \subset (E \cup \bigcup E)$. On en déduit $(x, y) \in \mathcal{P}(E \cup \bigcup E)$, d'où $f \in \mathcal{P}(\mathcal{P}(E \cup \bigcup E))$, et $\prod E \subset \mathcal{P}(\mathcal{P}(E \cup \bigcup E))$.

Remarques. On a $\prod \emptyset = \{\emptyset\}$: l'unique élément de $\prod \emptyset$ est l'unique fonction de domaine vide. Par contre, on a aussi $\prod\{\emptyset\} = \emptyset$ et, plus généralement $\prod E = \emptyset$ si $\emptyset \in E$. La réciproque est vraie si E est un ensemble fini; on peut d'ailleurs être plus précis dans ce cas.
Soit $E = \{e_1, \ldots, e_n\}$; si l'un des e_i est vide, alors $\prod E$ est vide. Si aucun e_i n'est vide mais si l'un d'eux est infini, alors $\prod E$ est infini. Enfin, si tous les e_i sont finis, la taille de $\prod E$ est le produit (arithmétique) des tailles des e_i, ce qui justifie le nom de produit donné à l'opération symbolisée par $\prod$.
On étudiera au paragraphe 2.6.4 le cas où E est un ensemble infini d'ensembles non vides.

2.4.7. *Construction des ensembles infinis*

Les axiomes déjà introduits ne permettent pas la construction d'ensembles infinis. Par contre, si une classe infinie reçoit le statut d'ensemble, les axiomes permettront de définir d'autres ensembles infinis. La démarche naturelle consiste donc à décréter qu'une certaine classe infinie est un ensemble. Le seul candidat évident est la classe $\mathbf{N}$ des naturels, que l'on a identifiée au terme $\bigcap\{x : Ind(x)\}$. On introduit donc *l'axiome de l'infini*:

(2.7) $\quad ens(\bigcap\{x : Ind(x)\})$.

Une meilleure connaissance de l'ensemble $\mathbf{N}$ est nécessaire pour pouvoir exploiter cet axiome. En effet, si nous disposons d'un moyen d'*engendrer* les naturels, nous n'avons toujours pas de critère intrinsèque permettant de déterminer si un ensemble donné est un naturel.

2.4.8. *Adéquation, complétude et indépendance*

Les axiomes ou schémas d'axiomes déjà introduits sont

- les schémas d'extension, de séparation et de Church;
- les axiomes du vide, de la paire, de la réunion et de la puissance;
- l'axiome de remplacement;

- l'axiome de l'infini.

Un système axiomatique donne généralement lieu à trois questions.

1. Les axiomes sont-ils vrais dans le(s) modèle(s) qu'ils sont censés décrire ? (Adéquation)
2. Aucun axiome n'est-il superflu, c'est-à-dire conséquence logique des autres ? (Indépendance)
3. Le système ne peut-il pas être enrichi d'axiomes indépendants, vrais dans le(s) modèle(s) visé(s) ? (Complétude)

La première question n'est pas tout à fait pertinente ici, car les modèles visés ne sont pas connus. Notre position sera plutôt d'admettre comme "univers ensembliste" tout modèle de la théorie. Encore faut-il que le système axiomatique soit consistant, c'est-à-dire exempt de contradiction. L'élimination du paradoxe de Russell n'implique pas l'inexistence d'autres paradoxes ... Il est assez vain de vouloir établir formellement cette consistance : cela ne pourrait se faire que dans le cadre d'une théorie plus forte que la théorie des ensembles, dont la consistance serait donc davantage sujette à caution.[57]

On peut prouver (la consistance de la théorie de base étant admise) que la réponse à la troisième question est négative. On peut donc enrichir la théorie d'axiomes supplémentaires. Les axiomes déjà introduits sont les énoncés non contestés de la théorie, sans lesquels les objets modélisés n'auraient pas la qualité intuitive d'ensembles. Les axiomes que nous introduirons plus loin n'auront plus ce caractère incontestable.

Nous avons introduit les axiomes de manière aussi naturelle que possible, mais les derniers venus rendent superflus l'axiome du vide et l'axiome de la paire. Le vide, étant une sous-classe de l'ensemble des naturels, est nécessairement un ensemble. De même, si a et b sont des ensembles, on peut définir une fonction de domaine $\mathcal{PP}(\emptyset) = \mathcal{P}(\{\emptyset\}) = \{\emptyset, \{\emptyset\}\} = \{0, 1\}$ et de codomaine $\{a, b\}$, ce qui fait de la classe $\{a, b\}$ un ensemble, puisque la classe $2 = \{0, 1\}$ en est un.

2.5. Ordinaux et axiome de fondation

2.5.1. *Introduction*

La théorie des ordinaux généralise la notion de nombre naturel; elle est le premier et le principal exemple de l'utilité de la théorie des ensembles pour l'étude d'objets infinis. Elle doit aussi donner lieu à une définition plus commode de la classe $\mathbf{N}$, permettant d'obtenir des preuves du théorème provisoire et des théorèmes de Peano (2.3.3). On aura aussi un critère simple pour distinguer un naturel d'un autre ensemble.

[57] Ce résultat négatif, dû à K. Gödel, sort du cadre d'un chapitre introductif.

2.5.2. *Naturels*

L'ensemble $\mathbf{N}$ contient les nombres $0, 1, 2, \ldots$, où $0 = \emptyset$, $1 = \{0\}$, $2 = \{0, 1\}$ et ainsi de suite. Il ne comporte aucun autre élément, puisqu'il est le plus petit ensemble inductif. La relation d'appartenance dans $\mathbf{N}$ est censée modéliser la relation $<$. Si c'est bien le cas, la relation $\in$ dans $\mathbf{N}$ sera une relation d'ordre strict (antiréflexive, antisymétrique et transitive); elle sera en outre bien fondée: tout sous-ensemble non vide a de $\mathbf{N}$ admet un élément minimal, c'est-à-dire un élément b tel que $b \in a$ et $\forall x\,[x \in a \;\Rightarrow\; x \notin b]$. Enfin, la relation d'appartenance sera *totale*: si x et y sont deux naturels distincts, on a soit $x \in y$ soit $y \in x$. Tout ceci se résume en disant que $(\mathbf{N}, \in)$ est un bon ordre strict. Si n est un naturel, $(n, \in)$ sera aussi un bon ordre strict.

Un résultat important est la réciproque partielle de l'énoncé précédent. Si $(E, \preceq)$ est un bon ordre (fini) à n éléments, alors $(E, \preceq)$ est isomorphe à $(n, \leq)$. L'isomorphisme se construit aisément. Le minimum e_0 de E est associé à 0; le minimum e_1 de $E \setminus \{e_0\}$ est associé à 1, et ainsi de suite.

Remarques. La notion d'ensemble fini est très intuitive et a été abondamment utilisée dans ce chapitre; elle n'a pourtant pas été formalisée ni utilisée dans un énoncé formel définitif, c'est-à-dire pour lequel une version plus générale n'a pas été ou ne sera pas introduite ultérieurement.

Un bon ordre n'étant pas nécessairement fini, on généralisera la notion de naturel.[58] Plus précisément, si A est un ensemble bien ordonné, on attribuera à la classe des ensembles bien ordonnés isomorphes à A un représentant-type. De tels représentants seront appelés *ordinaux*, les ordinaux finis devant s'identifier aux naturels. Les naturels hériteront ainsi des propriétés des ordinaux. Ce projet implique au préalable une définition intrinsèque des ordinaux.

Nous avons vu, au chapitre I.9, que la théorie du bon ordre n'est pas une théorie du premier ordre. Cela n'empêche pas qu'il soit possible d'étudier la notion d'ensemble bien ordonné dans le cadre de la logique du premier ordre. Simplement, la théorie devra faire appel à d'autres notions que celle de relation d'ordre. C'est bien le cas de la théorie des ensembles qui, à partir de la notion d'appartenance, permet de définir une grande variété de concepts, parmi lesquels ceux qui nous intéressent ici.

2.5.3. *Définition des ordinaux*

Pour remplacer une définition (incommode) par une autre (plus commode), on cherche une série de conséquences (démontrées ou non) de la définition initiale. La conjonction de ces conséquences peut être prise comme nouvelle définition, à condition qu'elle soit équivalente à l'ancienne. Le procédé est intéressant si la série adéquate s'exprime avec concision. Nous allons l'appliquer

[58] Cela permettra aussi de préciser cette notion.

ici à la définition des ordinaux et des naturels (vus comme les ordinaux finis). Notre point de départ est l'ensemble des propriétés déjà évoquées des naturels. Rappelons qu'une égalité telle que $n' = \{0, 1, \ldots, n\}$, si éclairante soit-elle, ne constitue pas une définition des naturels; néanmoins, les conséquences de cette pseudo-définition, pour peu qu'elles soient représentées formellement, constituent une bonne base pour l'obtention d'une vraie définition.

- Les éléments d'un naturel sont aussi des sous-ensembles de ce naturel (théorème provisoire). On a
$$\forall x\,[x \in n \;\Rightarrow\; x \subset n]\,, \qquad \text{ou encore} \qquad \bigcup n \subset n\,.$$
Une classe a vérifiant $\bigcup a \subset a$ est dite *transitive*.[59]

- Etant donné deux naturels distincts, l'un des deux appartient à l'autre. Si n est un naturel, on a
$$\forall x \forall y\,[(x \in n \;\wedge\; y \in n \;\wedge\; x \neq y) \;\Rightarrow\; (x \in y \;\vee\; y \in x)]\,.$$
Une classe a est dite *alternée* lorsqu'elle est connectée par la relation d'appartenance, c'est-à-dire lorsqu'elle vérifie la formule obtenue en remplaçant n par a dans l'expression ci-dessus.

- Il reste à utiliser le fait que $\in$ est une relation bien fondée dans l'ensemble des naturels. Si b est une partie non vide de n, la formule
$$\exists y\,[y \in b \;\wedge\; \forall z\,(z \in b \;\Rightarrow\; z \notin y)]$$
exprime l'existence d'un élément minimal y de b.[60] Ceci se simplifie en
$$\exists y\,[y \in b \;\wedge\; b \cap y = \emptyset]\,.$$
Une classe b vérifiant cette formule est dite *régulière*; une classe dont toutes les parties non vides sont régulières est dite *ordinaire*.

Un naturel est donc un ensemble fini, alterné, ordinaire et transitif. Avant d'envisager de démontrer cette assertion et sa réciproque, il faut définir le concept d'ensemble fini. Cela pose un problème car, intuitivement, un ensemble est fini si on peut "compter" les éléments, le résultat du décompte étant précisément un naturel. Cette circularité conduit à postposer l'étude des ensembles finis; on introduit donc la définition d'un ordinal comme suit.

Un *ordinal* est un ensemble alterné, ordinaire et transitif.

La classe des ordinaux sera notée Ω.
Dans la suite de ce chapitre, on étudiera les ordinaux pour eux-mêmes (ils ont un intérêt intrinsèque, celui de modéliser la notion de bon ordre), mais aussi pour approfondir la notion de naturel. Plus précisément, il faudra étudier $\mathbf{N}$ comme partie de Ω; il faudra aussi trouver un critère opérationnel permettant de distinguer les naturels des autres ordinaux (s'ils existent), ce qui permettra aussi de préciser la notion d'ensemble fini.

[59] En effet, la formule peut se récrire $\forall x \forall y\,[(y \in x \;\wedge\; x \in a) \;\Rightarrow\; y \in a]$, ce qui évoque la transitivité habituelle.

[60] Rappelons que, dans le cas d'un ordre total, un élément minimal d'un sous-ensemble ne peut être que le minimum de ce sous-ensemble.

Il n'est pas inutile, avant de poursuivre notre étude, de rendre plus concrète la notion d'ordinal. En effet, même si la notion de bon ordre est relativement intuitive, il n'est pas toujours évident de "voir" la différence essentielle qui sépare un ordre total d'un bon ordre, puisque le second concept est plus restrictif que le premier dans le cas infini seulement.

Considérons d'abord le cas fini. Un ordinal fini est tout simplement un naturel (cela sera prouvé formellement plus loin). Un prototype d'ordinal fini est donc

$$9 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}.$$

Intuitivement et pratiquement, les ordinaux (finis) permettent de classifier les ensembles bien ordonnés (finis); le fait que la relation d'ordre strict sous-jacente soit l'appartenance est important d'un point de vue technique, mais l'est moins du point de vue conceptuel. Le choix de l'appartenance n'est imposé que par notre volonté de baser la théorie des ensembles sur cette seule relation d'appartenance. L'ensemble

$$\{a, b, c, d, e, f, g, h, i\},$$

muni de la relation d'ordre alphabétique, représente tout aussi adéquatement le "type ordinal" de n'importe quel ensemble totalement ordonné à neuf éléments.[61] En fait, toute relation de bon ordre convient, même si son usage est moins répandu que celui des ordres numérique et alphabétique. On peut même se dispenser, dans le présent contexte, de définir cette relation explicitement. Dans l'écriture

$$(a, b, c, d, e, f, g, h, i),$$

les parenthèses signalent que l'ensemble sous-jacent est ordonné; la relation d'ordre est indiquée par l'ordre dans lequel les éléments sont écrits. Pour nous résumer, toute suite finie sans répétition évoque un ensemble bien ordonné, ainsi que le type ordinal qui lui correspond.

Tout ceci n'apporte rien si on se limite au cas fini. La notion de type ordinal se confond alors avec celle de nombre naturel. En utilisant les points de suspension, éventuellement accompagnés de commentaires permettant de déterminer les éléments "résumés" par ces points de suspension, on peut évoquer sans ambiguïté des types ordinaux infinis relativement simples; voici quelques exemples :

[61] La notion de "type ordinal" est analogue à celle de produit cartésien; c'est une entité abstraite que l'on peut définir concrètement de plusieurs manières. Nous ne le ferons ici que dans le cas particulier des ensembles bien ordonnés. Le type ordinal d'un ensemble bien ordonné est (concrétisé par) l'unique ordinal qui lui est isomorphe.

1. $(0, 1, 2, 3, \ldots)$,
2. $(1, 2, 3, 4, 5, \ldots)$,
3. $(\ldots, -3, -2, -1, 0)$,
4. $(\ldots, -2, -1, 0, 1, \ldots)$,
5. $(0, 1, -1, 2, -2, 3, -3, \ldots)$,
6. $(0, 2, 4, \ldots\ 1, 3, 5, 7, \ldots)$,
7. $(-1, -2, -3, \ldots\ \ldots, 4, 3, 2, 1, 0)$.

Nous nous sommes limités à des exemples numériques pour rendre inutiles les commentaires; ce que représentent les points de suspension est clair dans chaque cas. Soulignons cependant que les points de suspension peuvent se rapporter à l'élément qui les précède ou à l'élément qui les suit; nous levons l'ambiguïté (dans les cas simples) en convenant de séparer par une virgule les points de suspension et l'élément auquel ils se rapportent. Cela ne signifie évidemment pas que cette notation permet de formaliser la notion de type ordinal.

Deux séquences[62] représentent le même type ordinal si leurs éléments et leurs points de suspension sont semblablement disposés, ce qui permet d'établir entre les séquences une bijection, pas nécessairement unique, respectant la relation d'ordre. C'est le cas pour les séquences 1, 2 et 5.[63] Le type ordinal d'un ensemble totalement ordonné le caractérise "au renommage des éléments près". On voit immédiatement que les séquences 1, 3, 4, 6 et 7 représentent cinq types ordinaux distincts.

Certains types ordinaux (certains ensembles ordonnés, certaines séquences) admettent des suites infinies décroissantes; c'est le cas des types (représentés par les séquences) 3, 4 et 7.[64] Seuls les types ordinaux ne comportant pas de suites infinies décroissantes (tels celui représenté par les séquences 1, 2 et 5, et celui représenté par la séquence 6) correspondent à des ensembles bien ordonnés et se confondront avec les ordinaux. L'intérêt du concept d'ordinal par rapport à celui, plus général, de type ordinal, apparaît si on interprète la relation d'ordre comme signifiant "plus simple que". L'absence de suite infinie d'éléments de plus en plus simples dans un ensemble bien ordonné $(E, \preceq)$ garantit que les définitions récursives sur cet ensemble[65] ne sont pas des définitions "vicieuses"

---

[62] Nous évitons le mot "suite" ici, parce que l'objet dont il est question n'est pas parfaitement défini; en particulier, l'ensemble des index (§ 2.2.4) n'est pas défini. Une suite est une séquence dont les seuls points de suspension sont à la fin. A priori, une séquence représente (a pour domaine) un ensemble totalement ordonné quelconque, tandis qu'une suite représente un ensemble bien ordonné, par exemple un naturel (suite finie) ou un autre ordinal (suite infinie).

[63] La bijection entre les séquences 1 et 2 est $\{(0, 1), (1, 2), \ldots\}$; celle entre les séquences 2 et 5 est $\{(1, 0), (2, 1), (3, -1), (4, 2), \ldots\}$; remarquons aussi que les séquences $(0, 1, 2, \ldots)$ et $(0, 1, 2, 3, \ldots)$, par exemple, doivent être regardées comme égales.

[64] Un exemple pour les types 3 et 4 est la suite $(-2, -4, -6, -8, \ldots)$; pour le type 7 on a l'exemple $(4, 8, 12, 16, \ldots)$.

[65] Tel le texte $f(x) =_{def} \mathcal{A}$, où $\mathcal{A}$ est une expression faisant intervenir l'ensemble $\{f(y) : y \in E \wedge y \prec x\}$.

ou "circulaires". Si par exemple on "définit" $f(x)$ en fonction de $f(x-1)$, la définition ne sera pas nécessairement vicieuse si le domaine de f doit être $\mathbf{N}$, mais le sera si ce domaine doit être $\mathbf{Z}$. Nous verrons que le raisonnement inductif et le schéma de von Neumann, introduits dans le cas particulier de l'ensemble des naturels, se généralisent à tout (ensemble isomorphe à un) ordinal.

Remarque. De prime abord, on pourrait penser que le type ordinal associé à la séquence 6 admet des suites infinies décroissantes. Ce n'est pas le cas; l'objet $(3, 1 \ldots, 6, 4, 2, 0)$, par exemple, est une séquence mais pas une suite.

2.5.4. *Propriétés des ordinaux*

Proposition 1. Dans une classe ordinaire et alternée a, la relation d'appartenance est un bon ordre strict.

Preuve. La relation d'appartenance est bien fondée sur a, donc antisymétrique, et totale. Elle est antiréflexive, car $x \in x$ impliquerait que $\{x\}$ n'admet pas d'élément minimal. Elle est aussi transitive. Si on a $x \in y$ et $y \in z$, on ne peut avoir $z \in x$ car, si tel était le cas, $\{x, y, z\}$ n'aurait pas d'élément minimal. On a donc nécessairement $x \in z$.

Remarque. La classe a vérifie $\forall x\,(x \in a \Rightarrow x \notin x)$; on en déduit immédiatement $a \notin a$.

Proposition 2. Tout élément b d'un ordinal a est un ordinal et est strictement inclus dans a.

Preuve. L'élément b est une partie de a (transitivité de a), distincte de a puisque $a \notin a$; l'ensemble b est ordinaire et alterné puisqu'il est un sous-ensemble d'un ensemble ordinaire et alterné. Il reste à montrer que b est transitif. Soit $c \in b$ et $d \in c$; il faut prouver $d \in b$. On observe d'abord que b, c et d sont éléments de a. En effet, l'ordinal a est transitif donc, de $b \in a$, on déduit $b \subset a$, $c \in a$, $c \subset a$ et $d \in a$. La classe a étant ordonnée par la relation d'appartenance, on déduit $d \in b$ par transitivité, à partir de $d \in c$ et $c \in b$, ce qui achève la preuve.

Proposition 3. Tout ensemble transitif b strictement inclus dans un ordinal a est un élément de a.

Preuve. La partie $a \setminus b$ de a est régulière; il existe donc un élément $c \in (a \setminus b)$ tel que $c \cap (a \setminus b) = \emptyset$ (c est le minimum de $a \setminus b$). Il est suffisant de prouver $c = b$. Tout d'abord, $c \in a$ implique $c \subset a$ et, par construction, $c \subset b$. D'autre part, si $d \in b$, alors on a $d \in c$, $d = c$ ou $c \in d$. Le deuxième cas est impossible car il impliquerait $c \in b$; le troisième cas est impossible car il impliquerait $(c \in d \wedge d \in b)$ et donc $c \in b$. On a donc $(d \in b \Rightarrow d \in c)$ pour tout d, c'est-à-dire $b \subset c$, et $b = c$.

Proposition 4. La classe Ω est alternée.

Preuve. Soient $a, b \in \Omega$; si $a \notin b$ et $a \neq b$, alors $a \not\subset b$ (prop. 3). L'ensemble $a \cap b$ est donc transitif et strictement inclus dans a, et un élément de a. De $a \cap b \in a$ et $a \cap b \notin a \cap b$, on déduit $a \cap b \notin b$, d'où on tire successivement

$a \cap b = b$, $b \subset a$ et $b \in a$.
Remarque. Les relations d'appartenance et d'inclusion stricte sont égales dans $\Omega \times \Omega$.

Proposition 5. Toute classe non vide d'ordinaux possède un $\subset$-minimum (qui est aussi un $\in$-minimum).
Preuve. Soient $K \subset \Omega$ et $a \in K$. Si $a \cap K$ est vide, alors a est le minimum de K. Sinon, la classe $a \cap K$, étant régulière, admet un élément minimal b, tel que $b \cap a \cap K = \emptyset$. Cet élément est le minimum de K. On a, d'une part, $b \in K$ et, d'autre part, $b \in a$, $b \subset a$, $b \cap a = b$, $b \cap K = b \cap a \cap K = \emptyset$.
Remarque. Tout ensemble transitif d'ordinaux est un ordinal.

Proposition 6. La classe Ω est propre, ordinaire, alternée et transitive; elle est aussi inductive.
Preuve. Il est clair que la classe Ω est ordinaire (prop. 5), alternée (prop. 4) et transitive (prop. 2); si elle était un ensemble, elle serait un ordinal et on aurait $\Omega \in \Omega$, ce qui est impossible. Enfin, $0 \in \Omega$ et, si $x \in \Omega$, alors $x \subset \Omega$ et $x' = x \cup \{x\} \subset \Omega$; x' est un ensemble transitif d'ordinaux, d'où $x' \in \Omega$.
Remarque. Dans la théorie naïve, la classe des ordinaux était paradoxale, comme la classe de Russell, et pour une raison analogue: non seulement la formule $\Omega \in \Omega$ était inacceptable (comme dans la théorie axiomatique), mais la formule $\Omega \notin \Omega$ l'était également. En effet, en l'absence de distinction entre ensemble et classe propre, la classe Ω vérifiait tous les critères requis d'un ordinal (élément de Ω) quelconque.

Proposition 7. Tout ordinal x vérifie les formules suivantes:

$$\forall y\,[y \in x' \equiv (y \in x \vee y = x)]\,,$$
$$\forall y\,[y \in x \equiv (y' \in x \vee y' = x)]\,.$$

Preuve. La première formule est une conséquence de l'équivalence $y \in \{x\} \equiv y = x$, valide pour tout ensemble x. La seconde se démontre comme suit. Si on a $y \in x$, on a aussi $y \subset x$ et $y \cup \{y\} \subset x$, c'est-à-dire $y' \subset x$. Comme y' est transitif, on a $y' \in x \vee y' = x$. Réciproquement, on a $y' = x \Rightarrow y \in x$ et $y' \in x \Rightarrow y \in x$.

Proposition 8. Si x est un ordinal, alors $\bigcup x$ est un ordinal; en outre, on a $\bigcup x \subset x = \bigcup x'$.
Preuve. On voit facilement que $\bigcup x$ est une partie transitive de x. De $x \in x'$, on déduit $x \subset \bigcup x'$; réciproquement, s'il existe z tel que $y \in z \wedge z \in x'$, alors on a $y \in x$, d'où $\bigcup x' \subset x$.

Principe d'induction ordinale.

$$\frac{\forall x\,[(x \in \Omega \wedge \forall y\,[y \in x \Rightarrow P(y)]) \Rightarrow P(x)]}{\forall x\,[x \in \Omega \Rightarrow P(x)]}$$

Preuve. Supposons vérifié l'antécédent de la règle. Si la classe $K = \{x : x \in \Omega \wedge \neg P(x)\}$ n'est pas vide, elle admet un minimum m. Cet élément vérifie, d'une part, $\neg P(m)$ et, d'autre part, $m \in \Omega \wedge \forall y\,[y \in m \Rightarrow P(y)]$. L'antécédent

du principe permet de déduire $P(m)$ et donc une contradiction.

Représentation des ordinaux par des séquences. Informellement, le lecteur aura peut-être deviné qu'une séquence ne peut représenter un ordinal si elle comporte des points de suspension se rapportant à l'élément qui les suit. Informellement toujours, si on exclut cette possibilité, on distingue trois sortes de séquences susceptibles de représenter des ordinaux. On a l'unique séquence sans élément, les séquences se terminant par un élément, et les séquences se terminant par des points de suspension. Cette classification peut se formaliser comme suit.

Classification des ordinaux. La classe Ω se partitionne en le singleton $\{0\}$, la classe *Suc* des ordinaux *successeurs* et la classe *Lim* des ordinaux *limites*. Les classes *Suc* et *Lim* sont définies par les égalités suivantes :

$$Suc = \{x : x \in \Omega \wedge \exists\, y\,(x = y')\}\,;$$
$$Lim = \{x : x \in \Omega \setminus \{0\} \wedge \neg\exists\, y\,(x = y')\}\,.$$

Remarque. On peut écrire "$\exists!y\,(x = y')$" au lieu de "$\exists\, y\,(x = y')$".

Intuitivement, on sait que 0 est l'ensemble vide, le premier ordinal. De même, un ordinal successeur "suit" immédiatement un autre ordinal. La notion d'ordinal limite semble a priori moins intuitive; un ordinal est limite s'il n'est pas 0 ou successeur. On peut appréhender de façon plus positive, et plus intuitive, la notion d'ordinal limite. Rappelons d'abord que la motivation principale de la notion même d'ordinal est la volonté d'obtenir des "nombres infinis" permettant de classifier des ensembles infinis (ordonnés ou non). Il semble inévitable, au moins a posteriori, que le premier "nombre infini" permette de compter le premier ensemble infini, celui des naturels. Tout naturel n est un ensemble à n éléments, représentatif de tous les n-uplets. Le premier nombre infini aura autant d'éléments que l'ensemble des naturels, ce qui incite à identifier ce nombre à cet ensemble. Il est clair que l'ensemble des naturels n'est le successeur d'aucun ensemble, ce qui impose d'emblée la considération d'ordinaux d'un troisième type. Pour caractériser intuitivement les ordinaux de ce nouveau type, on peut répertorier les propriétés distinguant les naturels (successeurs) de l'ensemble des naturels. Tout d'abord, tout naturel $n = \{0, \ldots, n-1\}$ est le successeur de $\bigcup n = (0 \cup \cdots \cup n-1) = n-1$. L'ensemble des naturels n'aura pas cette propriété, et on observe immédiatement qu'il s'identifie à la réunion de ses éléments; on observe aussi qu'il n'a pas de maximum; si l'ensemble des naturels contient x, il contient aussi le successeur de x. Notons aussi que, dans le cas des suites d'ensembles $\subset$-croissantes, la notion intuitive de limite est adéquatement formalisée par la notion de réunion. Dire d'un ordinal qu'il s'identifie à la réunion de ses éléments, c'est dire qu'il est la limite de la suite des ordinaux plus petits que lui. Ces considérations justifient le terme "ordinal limite"; elles sont formalisées dans le théorème suivant.[66]

[66]La première propriété distinguant l'ensemble des naturels de ses éléments est son

Proposition 9. Les trois classes définies ci-dessous sont égales à Lim:

$$Lim_1 = \{x : x \in \Omega \setminus \{0\} \wedge \forall y (y \in x \Rightarrow y' \in x)\},$$
$$Lim_2 = \{x : x \in \Omega \setminus \{0\} \wedge x = \bigcup x\},$$
$$Lim_3 = \{x : x \in \Omega \setminus \{0\} \wedge x \subset \bigcup x\}.$$

Preuve. Le résultat se déduit de l'équivalence des formules suivantes. Ces formules, dans lesquelles x désigne un ordinal non nul, sont vraies si x est limite, et fausses si x est successeur.

$$\neg\exists y\,(x = y'),$$
$$\forall y (y \notin \Omega \vee x \in y' \vee y' \in x),$$
$$\forall y (y \notin \Omega \vee x \in y \vee x = y \vee y' \in x),$$
$$\forall y (y \notin x \vee y' \in x),$$
$$\forall y (y \in x \Rightarrow y' \in x),$$
$$\forall y (y \in x \Rightarrow (y \in y' \wedge y' \in x)),$$
$$\forall y (y \in x \Rightarrow \exists z (y \in z \wedge z \in x)),$$
$$\forall y (y \in x \Rightarrow y \in \bigcup x),$$
$$x \subset \bigcup x,$$
$$x = \bigcup x.$$

On peut à présent relier les naturels aux ordinaux et, enfin, donner une définition intrinsèque du concept de naturel. On observe d'abord que les ordinaux limites sont exactement les ordinaux inductifs. Cela implique que l'ensemble **N**, en sa qualité de plus petite classe inductive, est inclus dans tout ordinal limite. On introduit alors la classe

$$\omega =_{def} \{x : x \in \Omega \setminus Lim \wedge \forall y (y \in x \Rightarrow y \notin Lim)\}.$$

Cette classe d'ordinaux est non vide ($0 \in \omega$) et transitive: si $z \in x$ et $x \in \omega$, alors $z \subset x$, $z \in \Omega \setminus Lim$ et $z \in \omega$. La classe ω est aussi inductive; si ω est un ensemble, ce sera aussi un ordinal limite, nécessairement le plus petit.

L'ensemble **N** des naturels s'identifie à ω. D'une part, **N** est inclus dans ω puisque ω est inductif et, d'autre part, ω est inclus dans **N**; en effet, si $\omega \setminus \mathbf{N}$ n'est pas vide, son minimum est nécessairement un ordinal successeur x'. Or, si $x' \in \omega \setminus \mathbf{N}$, on a aussi $x \in \omega \setminus \mathbf{N}$, d'où une contradiction. L'ensemble des naturels est donc le plus petit ordinal limite. (Comme ordinal, **N** est transitif, ce qui valide le théorème provisoire introduit au paragraphe 2.3.3.) On a donc la définition suivante.

Un *naturel* est un ordinal non limite ne contenant aucun ordinal limite.

Il est maintenant possible de définir le concept d'ensemble fini. Un ensemble est *fini* s'il est le codomaine d'une fonction dont le domaine est un naturel.

Proposition 10. Si A est une sous-classe non vide de Ω, alors $\bigcap A$ est le minimum de A, et $\bigcup A$ est la meilleure borne supérieure de A.

---

caractère infini ... mais nous ne savons pas encore exprimer intrinsèquement cette propriété pourtant capitale. Nous reviendrons sur ce point plus loin.

Preuve. Soit A une classe non vide d'ordinaux. Vu la proposition 5, la classe A est ordinaire; comme elle est incluse dans Ω, elle est aussi alternée (prop. 4). Dans A, la relation $\in$ est donc un bon ordre strict (prop. 1) et la relation $\subset$ est donc un bon ordre sur A. Ceci implique que $\bigcap A$ et $\bigcup A$ sont respectivement la meilleure borne inférieure et la meilleure borne supérieure de A. La borne inférieure est nécessairement atteinte (prop. 5).
Remarques. On n'a pas toujours $\bigcup A \in A$. En fait, quatre cas sont possibles; $\bigcup A$ peut être 0 (borne supérieure atteinte, $A = \{0\}$), un ordinal successeur (borne atteinte), un ordinal limite (borne atteinte ou non), ou enfin la classe propre Ω elle-même (borne non atteinte). Dans tous les cas, $\bigcup A$ est ordinaire, alterné et transitif.
On observe aussi que Ω est la seule classe propre transitive d'ordinaux.
Rappelons qu'une *suite* est une fonction dont le domaine est une classe bien ordonnée. On appelle *suite ordinale* une fonction dont le domaine est un ordinal ou la classe des ordinaux.

2.5.5. *Principe d'induction ordinale et récursivité ordinale*

Le principe de récurrence est un schéma de raisonnement permettant de prouver des énoncés du type "la formule $P(n)$ est valide pour tout naturel n". Une version formelle de ce principe est la règle d'inférence suivante :

$$\frac{P(0) \wedge \forall n\,[P(n) \Rightarrow P(n+1)]}{\forall n\,P(n)}$$

L'expression $P(x)$ désigne ici toute formule où x est une variable libre. Les quantifications sur n sont restreintes à l'ensemble des naturels mais, conformément à un usage fréquent, cette restriction est implicite.

Une forme plus générale (en apparence seulement) est la règle de "récurrence complète" :

$$\frac{\forall n\,[\forall m\,(m < n \Rightarrow P(m)) \Rightarrow P(n)]}{\forall n\,P(n)}$$

Cette règle peut se déduire de la précédente, et réciproquement.

La règle d'induction ordinale est formellement la même, mais la quantification est maintenant restreinte à une classe Γ transitive, ordinaire et alternée (on a donc $\Gamma = \Omega$ ou $\Gamma \in \Omega$) plutôt qu'à l'ensemble $\mathbf{N}$; on écrit

$$\frac{\forall \alpha\,[\forall \beta\,(\beta \in \alpha \Rightarrow P(\beta)) \Rightarrow P(\alpha)]}{\forall \alpha\,P(\alpha)}$$

Il est parfois intéressant de définir récursivement des suites ordinales quelconques et non seulement des suites sur $\mathbf{N}$. Le principe d'induction ordinale permet de valider une forme plus générale du théorème de von Neumann.

Théorème 2.6. Soit Γ une classe ordinaire, alternée et transitive, soit A une classe non vide, soit Φ la classe des fonctions dont le domaine est un élément

de Γ et le codomaine est inclus dans A. Si F est une fonction de Φ dans A, alors il existe une et une seule fonction h de Γ dans A telle que

$$h(r) = F(h|_r)\,, \text{ pour tout } r \in \Gamma\,.$$

Remarques. Le "théorème" de von Neumann est en fait un schéma de théorèmes, de même que le principe de séparation est un schéma d'axiomes. Dans ce dernier cas, on obtient un axiome pour chaque formule $P(x)$ à une variable libre. Dans le cas présent, on obtient un théorème pour chaque formule (ou ensemble fini de formules) définissant des objets Γ, A et F respectant les conditions de l'énoncé.
Les fonctions dont le domaine est un élément de Γ sont des ensembles, ce qui permet l'évocation de la classe Φ. Par contre, la fonction F peut être une classe propre et il n'est pas possible de regrouper dans une classe les fonctions qui vérifient les conditions imposées à F. La fonction h sera une classe propre si Γ est une classe propre. Par contre, $h|_r$ et $h(r)$ seront toujours des ensembles.

Preuve. Elle généralise celle du théorème 2.2. On considère la suite ordinale de fonctions (h_τ) telle que $h_0 = \emptyset$, $h_{\tau'} = h_\tau \cup \{(r, F(h_\tau))\}$, pour tout $r \in \Gamma$, et $h_\ell = \bigcup\{h_r : r \in \ell\}$, pour tout $\ell \in \Gamma \cap Lim$. On a la propriété suivante : pour tout $r \in \Gamma$, h_r est une fonction de domaine r et, pour tout $s \in r$, on a $h_s = h_r|_s$. Ceci se démontre par l'absurde. Soit ρ le plus petit ordinal pour lequel la propriété est fausse. On a $\rho \neq 0$ car $h_0 = \emptyset$ vérifie la propriété. On considère séparément le cas où ρ est un ordinal successeur et celui où ρ est un ordinal limite.
Si ρ est successeur de l'ordinal σ, on a $h_\rho = h_\sigma \cup \{(\sigma, F(h_\sigma))\}$. Comme h_σ est une fonction de domaine σ, h_ρ est une fonction de domaine $\rho = \sigma \cup \{\sigma\}$. De plus, pour tout $\tau \in \rho$, on a $h_\tau = h_\rho|_\tau$. On distingue deux cas. Si $\tau \in \sigma$, alors $h_\tau = h_\sigma|_\tau = h_\rho|_\tau$ puisque h_σ et h_ρ ne diffèrent que par un couple dont l'origine σ n'appartient pas à τ. En outre, si $\tau \notin \sigma$, on a $\tau = \sigma$ et $h_\sigma = h_\rho|_\sigma$ par construction.
Si ρ est un ordinal limite, on a $h_\rho = \bigcup\{h_\tau : r \in \rho\}$, et le domaine est bien ρ. On voit que h_ρ est une fonction, car $(r, x) \in h_\rho$ si et seulement si $h_{r'}(r) = x$. De plus, pour tout $\tau \in \rho$, on a $h_\tau = h_{\tau'}|_\tau = h_\rho|_\tau$.
La propriété est donc vraie pour ρ (limite ou successeur), d'où une contradiction.
On définit maintenant $h =_{def} \bigcup\{h_{r'} : r \in \Gamma\}$. L'objet h est un ensemble de couples (relation binaire). Le domaine de la relation h est Γ puisque, pour tout $r \in \Gamma$, h inclut un élément dont le domaine contient r. Cette relation est une fonction et $h(r)$ est égal à $h_p(r)$, quel que soit p tel que $r \in p$.
Il reste à montrer que h est la seule fonction pour laquelle le schéma de von Neumann est vérifié. Soit, par l'absurde, une autre solution h' et soit ρ le plus petit ordinal tel que $h'(\rho) \neq h(\rho)$. Cet ordinal n'est pas 0 car $h'(0) = F(0) = h(0)$. On a donc $h'(\rho) = F(h'|_\rho)$ et $h(\rho) = F(h|_\rho)$. Comme les fonctions h et h' sont égales pour tout $\tau \in \rho$, on a $h'|_\rho = h|_\rho$, d'où une contradiction.

Remarque. Cette technique de définition ordinale est d'usage fréquent. Elle permet par exemple la démonstration constructive du théorème suivant, qui formalise l'un des intérêts majeurs de la notion d'ordinal.

Théorème 2.7. Tout ensemble bien ordonné E est isomorphe à un ordinal.
Preuve. On utilise le schéma de von Neumann, instancié comme suit :

$$\begin{aligned}
&\Gamma = \Omega\,,\\
&A = \mathcal{P}(E)\,,\\
&h(0) = E\,,\\
&h(x') = (\text{si } h(x) = \emptyset \text{ alors } \emptyset \text{ sinon } h(x)\setminus\{\min(h(x))\})\,,\\
&h(\ell) = \textstyle\bigcap_{y\in\ell} h(y)\,.
\end{aligned}$$

Dans ces formules, x désigne un ordinal quelconque et ℓ désigne un ordinal limite. On prouve facilement que $h(x)$ est inclus dans E et que si $x \leq y$ on a $h(y) \subset h(x)$. Rappelons aussi que, par définition du bon ordre, toute partie non vide de E admet un minimum. L'ordinal cherché est le minimum de la classe $\{r : r \in \Omega \wedge h(r) = \emptyset\}$; l'isomorphisme est la fonction qui, à tout $x \in r$, associe $\min(h(x)) \in E$.

Remarques. Toute classe propre bien ordonnée est isomorphe à la classe Ω des ordinaux, et réciproquement. Notons aussi qu'entre deux classes bien ordonnées existe au plus un isomorphisme; en effet, il n'existe pas d'isomorphisme entre deux ordinaux distincts, et le seul isomorphisme d'un ordinal sur lui-même est l'identité. Ces résultats ne s'étendent pas aux classes totalement ordonnées. Par exemple, pour tout entier $a \in \mathbf{Z}$, la fonction f qui à tout entier z associe l'entier $z + a$ est un *automorphisme* de $\mathbf{Z}$, c'est-à-dire un isomorphisme de $\mathbf{Z}$ sur lui-même.

2.5.6. *Applications du théorème de von Neumann*

Le théorème de von Neumann permet de définir l'addition, la multiplication et l'exponentiation ordinales, dont les opérations arithmétiques correspondantes sur les naturels sont des restrictions. Ces opérations se définissent par cas, selon la classification des ordinaux introduite au paragraphe 2.5.4. Nous utilisons des notations informelles, tout en sachant qu'on peut les rendre formelles. Dans ce qui suit, x et y sont des ordinaux quelconques et z est un ordinal limite.

$$\begin{aligned}
&x + 0 =_{def} x\,,\\
&x + y' =_{def} (x + y)'\,,\\
&x + z =_{def} \textstyle\bigcup\{x + y : y \in z\}\,.
\end{aligned}$$

$$\begin{aligned}
&x.0 =_{def} 0\,,\\
&x.y' =_{def} (x.y) + x\,,\\
&x.z =_{def} \textstyle\bigcup\{x.y : y \in z\}\,.
\end{aligned}$$

$$x^0 =_{def} 1,$$
$$x^{y'} =_{def} (x^y).x,$$
$$0^z =_{def} 0,$$
$$x^z =_{def} \bigcup\{x^y : y \in z\} \quad (x > 0).$$

Les opérations ordinales ne conservent pas toutes les propriétés des opérations arithmétiques. Par exemple, l'addition ordinale n'est pas commutative; on a $\omega + 1 \neq \omega$, alors que $1 + \omega = \omega$. Cela traduit le fait que, dans la théorie de l'ordre total, les ensembles $\{-1\} \cup \omega$ et ω sont isomorphes (l'isomorphisme est $x \mapsto x+1$, alors que les ensembles ω et $\omega \cup \{+\infty\}$ ne le sont pas (seul le second admet un maximum).

Un moyen informel d'appréhender les opérations ordinales est de les traduire en termes de séquences. Une séquence modélisant $\alpha + \beta$ s'obtient en faisant suivre une séquence modélisant α d'une séquence modélisant β, avec insertion d'une virgule si la première séquence se termine par un élément, c'est-à-dire si α est un ordinal successeur.

Une séquence modélisant $\alpha \,.\, \beta$ s'obtient en remplaçant chaque élément d'une séquence modélisant β par une séquence modélisant α. On infère ainsi que $2.\omega = \omega$, alors que $\omega.2$ en est distinct, et représenté notamment par la séquence $(0, 2, 4, \ldots\ 1, 3, 5, \ldots)$.

L'exponentiation est moins facile à visualiser et nous ne l'aborderons que par le biais d'exemples simples. Considérons la séquence

$$(1, 2, 4, 8, 16, \ldots\ 3, 6, 12, \ldots\ 9, 18, 36, \ldots\ldots).$$

Apparaissent dans cette séquence tous les naturels strictement positifs admettant 2 et/ou 3 comme seuls facteurs premiers. Le nombre 2^p3^q précède le nombre 2^r3^s si et seulement si on a $q < s$ ou $q = s$ et $p < r$. Cette séquence représente l'ordinal $\omega^2 = \omega.\omega$; des séquences équivalentes sont

$$((0,0), (1,0), (2,0), \ldots\ (0,1), (1,1), (2,1), \ldots\ (0,2), (1,2) \ldots\ldots)$$

et

$$(0, 1, 2, \ldots\ \omega, \omega+1, \omega+2, \ldots\ \omega.2, \omega.2+1, \ldots\ldots).$$

On voit facilement comment obtenir une séquence modélisant ω^n, pour tout naturel n, et une séquence modélisant ω^ω. En particulier, on peut arranger les naturels, sans répétition ni omission, en une séquence modélisant ω^ω. Ceci se généralise; on peut définir une "méta-exponentiation" f telle que $f(0) = 1$ et $f(n+1) = \omega^{f(n)}$; l'ordinal $f(n)$ reste représentable par une séquence dont les éléments sont des naturels distincts, et il en est de même de $f(\omega)$ défini comme $\bigcup_n f(n)$.

Remarque. Si a et b sont des ensembles, la notation a^b est parfois utilisée pour désigner l'ensemble des fonctions dont le domaine est b et dont le codomaine est inclus dans a. Pour éviter toute confusion avec l'exponentiation ordinale, il est préférable de désigner l'ensemble des fonctions de b dans a par l'expression $(b \to a)$. C'est d'autant plus prudent que les deux notions sont apparentées

sans être équivalentes: si a et b sont des naturels, les ensembles $(b \to a)$ et a^b peuvent être mis en bijection;[67] plus simplement, l'ensemble $(b \to a)$ comporte exactement a^b éléments. Cette propriété ne se généralise pas au cas infini. Par exemple, on verra que les ensembles $(\omega \to \omega)$ et ω ne sont pas équipotents, alors que les ensembles ω^ω et ω le sont.
Ce dernier point est facile à vérifier. Remarquons d'abord que, pour tout naturel n, l'ordinal ω^n est isomorphe à l'ensemble des n-uplets ordonnés de naturels, muni de la relation d'ordre lexicographique; en effet, un élément de ω^n s'écrit, d'une seule façon, sous la forme $\omega^{n-1}.a_1 + \omega^{n-2}.a_2 + \cdots + \omega.a_{n-1} + a_n$. L'ordinal ω^ω est, par définition, la réunion des ordinaux ω^n. Un élément de ω^ω est donc une suite finie de naturels, et l'ensemble des suites finies de naturels est dénombrable. Rappelons qu'un ensemble est dénombrable s'il est fini ou équipotent à $\mathbf{N}$ (ou ω). Si les ordinaux α et β sont dénombrables, les ordinaux $\alpha+\beta$, $\alpha.\beta$ et α^β le sont également. On reviendra sur la dénombrabilité plus loin.

L'induction ordinale permet aussi de justifier la validité des programmes fonctionnels, dans des cas où l'application de l'induction naturelle est difficile.

Rappelons d'abord la définition classique des coefficients binomiaux, évoquée au paragraphe 2.3.4.3:

$$\begin{aligned} C_n^0 &=_{def} 1, \text{ pour tout } n \in \mathbf{N}, \\ C_n^n &=_{def} 1, \text{ pour tout } n \in \mathbf{N}, \\ C_n^p &=_{def} C_{n-1}^p + C_{n-1}^{p-1}, \text{ pour tous } n \in \mathbf{N} \text{ et } p \in \{1, \ldots, n-1\}. \end{aligned}$$

L'induction ordinale s'applique; la classe Γ s'identifie à l'ordinal ω^2 (si l'on convient que $C(\omega.n + p)$ peut s'écrire C_n^p, et si on pose $C_n^p = 0$ si $n < p$). La justification se résume à observer que $\omega.(n-1) + p$ et $\omega.(n-1) + p - 1$ sont strictement inférieurs à $\omega.n + p$.

Dans ce cas élémentaire, le bénéfice de l'induction ordinale n'est pas évident, et l'induction naturelle était suffisante. On pouvait par exemple considérer la suite $((0,0),(1,0),(1,1),(2,0),(2,1),(2,2),(3,0),(3,1),(3,2),(3,3),(4,0),\ldots)$ et observer que, pour tout (n,p) apparaissant dans cette suite, le calcul de C_n^p ne requiert le calcul préalable de C que pour des valeurs précédentes (n',p') de l'argument.

Remarquons néanmoins que la formalisation de ce dernier point, quoique simple, est un peu fastidieuse. Bien plus, ce type de raisonnement ne s'adapte pas à tous les cas. Considérons par exemple la fonction d'Ackermann, "définie" par les égalités suivantes, valables pour tous $x, y \in \mathbf{N}$:

$$\begin{aligned} &A(0,y) = y+1, \\ &A(x+1,0) = A(x,1), \\ &A(x+1,y+1) = A(x,A(x+1,y)). \end{aligned}$$

[67] On dit qu'ils sont *équipotents*. Deux ensembles finis sont équipotents s'ils ont même nombre d'éléments.

On peut toujours, en considérant que $A(x,y)$ représente $A(\omega.x+y)$, observer que le domaine de A est ω^2 et que le calcul de $A(\alpha)$, pour tout $\alpha \in \omega^2$, requiert le calcul préalable de couples de la restriction $A|_\alpha$ exclusivement. Par contre, il n'est plus possible d'ordonner en une suite indexée par **N** tous les arguments possibles de A, en respectant la règle de précédence: (x,y) apparaît avant (x',y') seulement si le calcul de $A(x,y)$ ne requiert pas le calcul préalable de $A(x',y')$.

Remarque. Les théorèmes 2.6 et 2.7 permettent de justifier les définitions récursives de fonctions dont le domaine n'est pas un ordinal, mais n'importe quel ensemble bien ordonné; on a le résultat suivant.

Théorème 2.8. Soit (Γ, R) une classe bien ordonnée, soit A une classe non vide, soit F une fonction de $\Gamma \times V$ dans A; il existe une et une seule fonction h de Γ dans A telle que

$$h(x) = F(x, h|_{R^{-1}(x)\setminus\{x\}}), \text{ pour tout } x \in \Gamma.$$

Remarque. L'ensemble $R^{-1}(x)$ est, par définition, l'ensemble $\{y : (y,x) \in R\}$.

Preuve. Vu le théorème 2.7, il existe un isomorphisme (unique) θ entre (Γ, R) et $(\alpha, \in)$, où α est un ordinal approprié ou la classe Ω. Le théorème 2.6 permet de prouver l'existence et l'unicité d'une fonction g de domaine α et de codomaine A telle que $g(\gamma) = G(\gamma, g|_\gamma)$, pour tout $\gamma \in \alpha$ et pour toute fonction G de type approprié. Si on pose $G(\gamma,\phi) = F(\theta^{-1}(\gamma), \phi \circ \theta)$, pour toute fonction ϕ dont le domaine est un élément de α et dont le codomaine est inclus dans A, la fonction h cherchée est $g \circ \theta$.

Remarque. La technique de définition récursive des fonctions reste valide si le domaine n'est pas bien ordonné mais seulement bien fondé (§ 2.2.7); elle est d'usage courant en programmation fonctionnelle. Pour traiter cette nouvelle généralisation, on pourrait refaire la théorie de von Neumann dans un cadre plus vaste, qui sera brièvement introduit au paragraphe 2.5.7. Nous utiliserons un moyen plus expéditif, après avoir introduit l'axiome du choix (§ 2.6.4).

2.5.7. *Modèles ensemblistes et axiome de fondation*

La théorie des ordinaux est le couronnement de la théorie axiomatique des ensembles. D'une part, elle permet la définition des entiers naturels et des opérations arithmétiques; on a vu aux paragraphes 2.3.2-2.3.7 que, par ce biais, on pouvait modéliser tous les objets mathématiques usuels. D'autre part, les ordinaux permettent d'approcher avec précision la notion d'ensemble infini; il est clair que "la plupart" des ordinaux sont des ensembles infinis.[68] Les deux objectifs principaux de la théorie des ensembles sont donc atteints.

Nous avons développé l'axiomatisation de la théorie sur des bases intuitives, pour des raisons évidentes. Il est quand même nécessaire, à ce stade, de

[68] On verra à la section 2.6 que tout ensemble est équipotent à un ordinal.

s'interroger de manière plus abstraite sur les modèles (formels) dont la théorie rend compte. Plus précisément, le langage de la théorie des ensembles consiste en un symbole relationnel binaire, noté $\in$. On a vu qu'il était possible, au moyen de conventions et de mécanismes abréviatifs appropriés, de désigner formellement des classes particulières, telles la classe vide, les naturels, les ordinaux, la classe de Russell, la classe des ordinaux et la classe universelle. Conceptuellement, un *univers ensembliste*, c'est-à-dire un modèle de la théorie des ensembles telle qu'elle a été axiomatisée jusqu'ici, sera un graphe orienté infini. Les nœuds de ce graphe sont les classes et un arc de la classe a vers la classe b signifiera que, dans l'univers considéré, la classe a est un élément de la classe b.

On peut espérer que l'un de ces graphes se rapproche de l'univers ensembliste intuitif mais, cette dernière notion n'étant pas formellement définie, il est difficile d'étudier fructueusement cette question. Par contre, on peut essayer d'inventorier et de classifier les univers formels, en soulignant les propriétés intuitivement intéressantes. Rappelons tout d'abord que l'existence même d'un modèle, c'est-à-dire la consistance de la théorie, n'est en rien évidente. Bien au contraire, l'existence de paradoxes dans la théorie naïve rend celle-ci inconsistante, et il pourrait en être de même de notre théorie axiomatique.

Si nous admettons la consistance de la théorie — ce que nous avons fait depuis le début — nous devons cependant exclure l'unicité. Aucune théorie du premier ordre admettant un modèle infini n'est catégorique: c'est une conséquence du théorème de Löwenheim et Skolem dont une extension (théorème 9.12, volume I) affirme que toute théorie admettant un modèle infini admet nécessairement des modèles non isomorphes. Les divergences entre univers ensemblistes peuvent être importantes; par exemple, on peut concevoir un univers dont toute classe puisse être bien ordonnée, et aussi un univers dans lequel certaines classes ne jouiraient pas de cette propriété. La théorie des ensembles, comme la plupart des théories intéressantes, est incomplète: le langage de la théorie comporte des énoncés sans variables libres qui ne sont ni des théorèmes, ni des négations de théorèmes. Un résultat célèbre dû à Gödel montre que ces lacunes sont incontournables. Toute théorie du premier ordre qui contient l'arithmétique est essentiellement incomplète. Bien plus, sa consistance ne peut pas être établie dans le cadre de la théorie elle-même.

Même si les univers ensemblistes sont potentiellement nombreux et variés, ils présentent des caractéristiques communes, dont l'identification est intéressante. Un univers est, essentiellement, une relation binaire; il est donc naturel de commencer nos investigations par les propriétés classiques des relations binaires. Le résultat est décevant en ce qui concerne les propriétés "positives"; sur aucun univers la relation d'appartenance n'est réflexive, symétrique ou transitive. Par contre, l'appartenance semble bien antiréflexive et antisymétrique; plus généralement, on n'a pas rencontré jusqu'ici d'exemple de cycle pour l'appartenance, c'est-à-dire de classes $a_0, \ldots, a_n$ vérifiant les formules

$a_0 \in a_1, a_1 \in a_2, \ldots, a_{n-1} \in a_n$ et $a_n \in a_0$.

On conçoit pourtant que, a priori, la simple introduction d'un ensemble x tel que $x \in x$ n'induise pas nécessairement de contradiction.[69] On peut en fait montrer que des ensembles "circulaires" peuvent être introduits dans la théorie sans induire de contradiction, mais cette introduction n'a rien d'obligatoire. Ceci suggère l'adjonction d'un nouvel axiome, exprimant que la relation d'appartenance est bien fondée sur toute classe. On sait déjà que la relation d'appartenance est bien fondée sur toute classe d'ordinaux; l'*axiome de fondation* ne fera que généraliser cette propriété. On utilise la terminologie introduite au paragraphe 2.5.3; l'axiome s'énonce simplement :

Tout ensemble est ordinaire,

ou encore

Tout ensemble non vide est régulier.

Tout ensemble non vide c contient donc un élément a ne contenant aucun élément de c, c'est-à-dire un élément minimal pour la relation d'appartenance. Formellement, l'axiome de fondation s'écrit :

$$(2.8) \qquad \forall c\,(clp(c) \vee c = \emptyset \vee \exists a\,[a \in c \wedge a \cap c = \emptyset])\,.$$

L'axiome de fondation exclut le cas d'un ensemble x vérifiant $x \in x$: l'ensemble $\{x\}$ ne serait pas régulier puisque $x \cap \{x\} = \{x\} \neq \emptyset$. L'axiome exclut en fait tous les cycles, même infinis. Si un tel cycle existait dans un ensemble c, il formerait un sous-ensemble non régulier, n'admettant pas d'élément minimal pour la relation d'appartenance.

Il est intéressant d'observer que toute classe propre est également régulière. Si c est une classe propre contenant un élément a, on distingue deux cas. Si $c \cap a = \emptyset$, la proposition est évidente. Sinon, soit b un ensemble transitif tel que $a \subset b$. L'ensemble $b \cap c$ est régulier et contient un élément minimal d. L'intersection $d \cap c$ est vide car, si elle contenait un élément e, on aurait $e \in b \cap c \cap d$, et d ne serait pas minimal dans $b \cap c$.

Encore faut-il prouver que tout ensemble a est inclus dans un ensemble transitif b. On sait que a est transitif si $\bigcup a \subset a$. Si a est transitif, on choisit $b = a$; sinon, on adjoint à a les éléments de $\bigcup a$. Si $a \cup \bigcup a$ est transitif, on choisit $b = a \cup \bigcup a$, sinon, on itère le processus. Informellement, on a donc

$$b = a \cup \bigcup a \cup \bigcup \bigcup a \cdots$$

Formellement, on utilise le schéma de von Neumann pour définir b (la récurrence ordinaire suffit ici). L'axiome de fondation se simplifie donc en

$$(2.9) \qquad \forall c\,(c = \emptyset \vee \exists a\,[a \in c \wedge a \cap c = \emptyset])\,.$$

[69] L'objet paradoxal C vérifiait $C \in C$ dans la théorie naïve, mais la contradiction provenait du fait qu'il vérifiait aussi $C \notin C$; rappelons que dans la théorie axiomatique, C est une classe propre, ce qui exclut $C \in C$.

Un graphe Γ représentant un univers ensembliste sera nécessairement infini. La relation d'appartenance représentée par les arcs sera bien fondée et donc, en particulier, antiréflexive et antisymétrique. L'ensemble vide sera le minimum et les classes propres seront les éléments maximaux. Toute partie du graphe admettra au moins un élément minimal.

On représente naturellement toute classe c de l'univers par un fragment Γ_c du graphe Γ. Le fragment comportera tous les chemins (finis ou non) joignant le minimum $\emptyset$ à la classe c; les nœuds n'appartenant à aucun de ces chemins seront omis. La classe c est donc le maximum du graphe Γ_c.[70]

Le graphe Γ_c caractérise c; il n'est pas indispensable que les nœuds de ce graphe soient étiquetés: l'étiquetage peut toujours se reconstituer, de manière unique. Le graphe Γ_c est fini si et seulement si c est fini. A titre d'exemple, le graphe représentant l'ensemble $\{1,\{1\},3\}$ est donné à la figure 2.1. Le maximum de ce graphe est le nœud $\{1,\{1\},3\}$. Trois arcs arrivent à ce maximum, issus des nœuds 1, $\{1\}$ et 3.

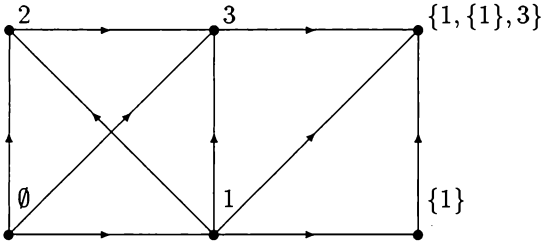

Figure 2.1 *Graphe* $\Gamma_{\{1,\{1\},3\}}$.

2.5.8. *Application de l'axiome de fondation*

La théorie des ordinaux de von Neumann permet notamment d'assimiler toute classe bien ordonnée (A,R) à une classe $(\alpha,\in)$, bien ordonnée pour la relation d'appartenance; α est Ω ou un élément de Ω. Pour étudier les bons ordres, il est suffisant d'étudier la relation d'appartenance sur les classes transitives, ordinaires et alternées.

De même, si (A,R) est un ensemble ordonné quelconque, il existe un ensemble B tel que (A,R) et $(B,\subset)$ soient isomorphes; l'inclusion est le prototype des relations d'ordre. Nous reviendrons en détail sur ce point au chapitre 5, mais l'isomorphisme est facile à construire. Pour tout $a \in A$, l'image pose $f(a)$ sera $R^{-1}(a)$, c'est-à-dire l'ensemble des éléments inférieurs à a (si A est un ensemble, $R^{-1}(A)$ l'est aussi, de même que la classe $B = f(A)$, incluse dans $\mathcal{P}(A)$.

[70] Plus précisément, c est le maximum de la classe E_c des nœuds de Γ_c, munie de la relation d'ordre $\in^*$, c'est-à-dire de la clôture réflexive et transitive de la relation d'appartenance.

L'axiome de fondation équivaut à admettre que, pour toute classe B, la classe ordonnée $(B, \in)$ est bien fondée. Ceci, juxtaposé aux énoncés précédents, suggère la question inverse. Si la classe strictement ordonnée (A, R) est bien fondée, existe-t-il une classe B telle que (A, R) et $(B, \in)$ soient isomorphes? Cela permettrait de ramener toutes les relations bien fondées (essentielles pour l'informaticien) à la relation d'appartenance.

Considérons d'abord un exemple simple: $A = \{a, b, ac, bac, cd, acd\}$ est un ensemble bien fondé pour la relation d'ordre R "est un sous-mot (strict) de". Le mot α est un sous-mot de β s'il existe un préfixe γ et un suffixe δ tels que $\beta = \gamma\alpha\delta$; le préfixe *ou* le suffixe peut être le mot vide λ. Le mot *bac* admet dans A les sous-mots a, b et ac, ce qui suggère son remplacement par l'ensemble $\{a, b, ac\}$; de même, ac admet le sous-mot a, ce qui suggère son remplacement par $\{a\}$. Plus généralement, une technique itérative de construction de B consiste à remplacer tout élément non R-minimal x par l'ensemble $R^{-1}(x)$. On a le schéma itératif suivant:

$$\begin{aligned} A_0 &= \{a, b, ac, bac, cd, acd\}, \\ A_1 &= \{a, b, \{a\}, \{b, ac\}, cd, \{ac, cd\}\}, \\ A_2 &= \{a, b, \{a\}, \{b, \{a\}\}, cd, \{\{a\}, cd\}\}. \end{aligned}$$

Deux itérations ont suffi ici pour passer de $A_0 = A$ à $A_2 = B$; en effet, seuls les éléments R-minimaux a, b et cd de A interviennent encore dans A_2.

Le théorème de von Neumann permet de formaliser ce processus de construction. Deux écueils peuvent se présenter. Tout d'abord, supposons que l'ensemble A ait contenu le mot supplémentaire *cdac*. Ce mot admet exactement les mêmes sous-mots stricts que le mot *acd*, déjà présent dans A. En conséquence, ces deux mots auront même représentant dans A_1 et dans $A_2 = B$, ce qui est inacceptable. La technique proposée s'applique donc seulement à une relation bien fondée R jouissant en plus de la propriété suivante: si $x \neq y$, alors $R^{-1}(x) \neq R^{-1}(y)$. Sous forme contraposée, on a $R^{-1}(x) = R^{-1}(y) \Rightarrow x = y$, pour tous $x, y \in A$. Dans le cas où R est la relation d'appartenance, on retrouve le principe d'extensionnalité. On qualifiera naturellement d'*extensionnelle* toute relation d'ordre (strict) bien fondé R vérifiant cette propriété.

L'extensionnalité de la relation R empêche toute "fusion" d'éléments au cours du processus itératif. Le second écueil concerne naturellement la terminaison de ce processus, mais on observe que les itérations $A_0, A_1, \ldots, A_n$ diffèrent si et seulement s'il existe une suite $(a_0, a_1, \ldots, a_n)$ d'éléments de A telle que $a_1 R a_0$, $a_2 R a_1, \ldots$, $a_n R a_{n-1}$. Dans l'exemple évoqué plus haut, une suite de ce type est (bac, ac, a). Le processus de construction de B au départ de A se termine si et seulement si la relation de départ est bien fondée.

Ce résultat n'étant pas utilisé plus loin, nous ne donnerons pas de preuve du *théorème de Mostowski*:

Théorème 2.9. La classe ordonnée (A, R) est bien fondée et extensionnelle si et seulement s'il existe une classe B telle que (A, R) et $(B, \in)$ sont isomorphes.

Remarques. Il ne faut pas confondre élément R-minimal et élément $\in$-minimal, c'est-à-dire élément fondamental; de plus, il ne faut pas perdre de vue l'ensemble de référence. Dans le cadre de notre exemple, a, b et cd sont les trois éléments R-minimaux de A. Si on ajoute à A le mot c, l'élément cd cesse d'être R-minimal. D'autre part, les éléments $\in$-minimaux de B sont aussi a, b et cd; cela n'implique naturellement pas qu'il s'agisse d'éléments fondamentaux, dont nous avons exclu l'existence dans le présent contexte (le seul élément fondamental est l'ensemble vide). En fait, a, b et cd représentent ici des ensembles distincts entre lesquels n'existe pas de relation d'appartenance; on peut choisir, par exemple, $a = \{1\}$, $b = \{2\}$ et $cd = \{3\}$.
Observons aussi qu'a priori, il semble envisageable de poursuivre le processus itératif de construction de B au delà des naturels et de l'ordinal ω, en utilisant la version ordinale du théorème de von Neumann. Cela poserait un problème sérieux: si la suite d'ensembles $(A_0, A_1, A_2, \ldots)$ n'est pas stationnaire, c'est-à-dire si on n'a pas $A_{n+1} = A_n$ pour un certain n, quel sens peut-on donner à la "limite" A_ω ? Les suites d'ensembles que nous avons rencontrées jusqu'ici étaient monotones pour la relation d'inclusion, ce qui permettait de poser $\lim A_n = \bigcup_n A_n$ (suite croissante) ou $\lim A_n = \bigcap_n A_n$ (suite décroissante). En dehors de ces cas, la notion de limite d'ensembles n'a pas de sens intuitif. Dans le même ordre d'idée, l'écriture $\{\{\ldots\{0\}\ldots\}\}$ n'a de sens et n'identifie un ensemble que si les points de suspension représentent un nombre fini d'accolades ouvrantes ou fermantes (le même dans les deux cas). Ceci est vrai même si on n'impose pas l'axiome de fondation. La théorie développée au paragraphe suivant permet, dans une certaine mesure, de donner une signification à ce type de notation.

2.5.9. *Ensembles non fondés*

Le paragraphe précédent suggère l'existence d'un lien entre ensembles formels et graphes. Avant de poursuivre, rappelons qu'un graphe $\Gamma = (V, R)$ se compose d'une classe V de points appelés *nœuds* et d'une relation binaire R sur V. Si $(x, y) \in R$, alors x est un *prédécesseur* de y et y est un *successeur* de x.[71] Un nœud minimal est un nœud n'admettant pas de prédécesseur; un nœud maximal n'admet pas de successeur. Si W est une partie de V, le sous-graphe correspondant est $(W, R \cap (W \times W))$. Un graphe est bien fondé si tout sous-graphe non vide admet un élément minimal. Si V_1 et V_2 sont disjoints, la réunion des graphes (V_1, R_1) et (V_2, R_2) est le graphe $(V_1 \cup V_2, R_1 \cup R_2)$. Un graphe est *connexe* s'il n'est pas la réunion de deux graphes non vides.

Nous avons vu que l'axiome de fondation permet de représenter un ensemble par un graphe connexe bien fondé. Ce résultat admet une réciproque partielle:

[71] Les graphes sont donc, a priori, orientés. Le graphe $\Gamma = (V, R)$ est parfois dit non orienté quand la relation R est symétrique.

on peut définir une vaste catégorie de graphes connexes bien fondés dont chaque membre représente un et un seul ensemble. Plus précisément, un graphe connexe bien fondé est *accessible* s'il admet un maximum, c'est-à-dire un nœud qui peut être atteint à partir de tous les autres.[72] La *décoration* d'un graphe accessible est la fonction qui à tout nœud du graphe associe un ensemble, en respectant les règles suivantes :

- A tout nœud minimal est associé l'ensemble vide.
- L'ensemble associé à un nœud non minimal est l'ensemble des ensembles associés aux prédécesseurs immédiats de ce nœud.

La condition d'accessibilité garantit l'existence et l'unicité de la décoration. L'ensemble associé à un graphe accessible est celui associé au nœud maximum de ce graphe. Un même ensemble peut être représenté par des graphes accessibles distincts. Un graphe accessible est *canonique* si sa décoration est une fonction injective. Tout ensemble E admet une et une seule représentation canonique Γ_E. Tout ensemble admet aussi une et une seule représentation arborescente, notée $\mathcal{T}_E$.

Les ensembles $0 = \emptyset$, $1 = \{0\}$, $\{1\} = \{\{0\}\}$, etc. admettent une et une seule représentation, à la fois canonique et arborescente. L'ensemble 2 admet deux représentations : la représentation canonique et la représentation arborescente, illustrées à la figure 2.2.

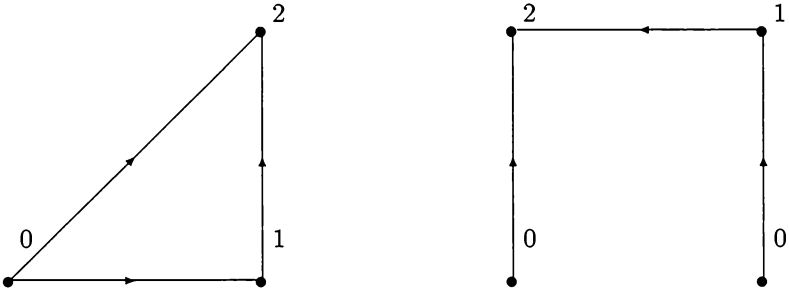

Figure 2.2 *Graphes Γ_2 (à gauche) et $\mathcal{T}_2$ (à droite).*

Etant donné un ensemble E, toute représentation de E s'obtient en "dépliant" la représentation canonique de E, c'est-à-dire en dédoublant certains nœuds dont plusieurs arcs sont issus. Le "dépliage" maximal donne lieu à la représentation arborescente. Nous ne formaliserons pas ici ce point mais, à titre d'exemple, la représentation arborescente (non décorée) de l'ensemble $\{1, \{1\}, 3\}$ est donnée à la figure 2.3.

[72] Cette notion peut se formaliser.

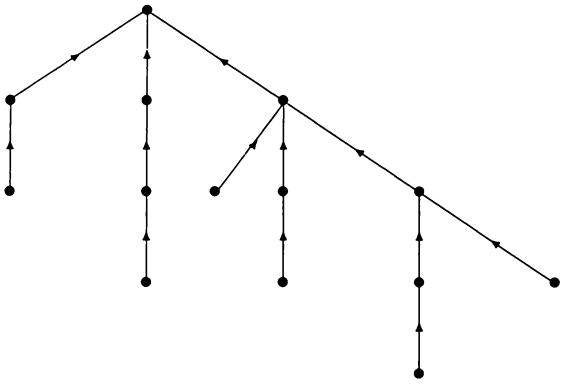

Figure 2.3 *Représentation arborescente de l'ensemble* $\{1, \{1\}, 3\}$.

L'axiome de fondation permet d'assimiler la théorie des ensembles à la théories des graphes bien fondés.[73]

Quoique les ensembles non bien fondés ou, plus classiquement, les classes non ordinaires, n'ont intuitivement guère d'intérêt, la correspondance avec la théorie des graphes montre que cet intérêt est réel. Si l'axiome de fondation est omis, voire remplacé par un énoncé qui le contredit, on peut espérer obtenir une "théorie ensembliste des graphes" non limitée aux graphes bien fondés.

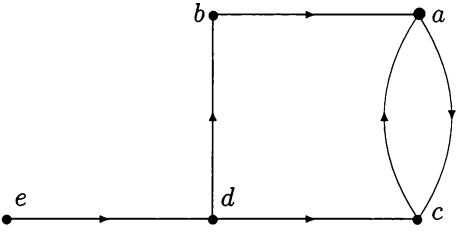

Figure 2.4 *Un graphe accessible non fondé.*

Considérons par exemple le graphe "non fondé" de la figure 2.4. Ce graphe reste accessible, en ce sens qu'un de ces nœuds, ici a, est distingué (le point le représentant est plus gros), et que tout nœud est origine d'un chemin

[73]La condition de connexité n'est guère restrictive; un graphe non connexe est simplement la juxtaposition d'une collection de graphes connexes. La condition d'accessibilité n'est pas très restrictive non plus. Si le graphe (V, R) n'est pas accessible, on le remplace par le graphe $(V \cup \{\alpha\}, R \cup (V \times \{\alpha\}))$, c'est-à-dire que l'on adjoint au graphe un nouvel élément qui sera le maximum.

aboutissant à a. Le graphe "représente" le système d'équations

$$\begin{aligned} a &= \{b, c\}\,, \\ b &= \{d\}\,, \\ c &= \{d, a\}\,, \\ d &= \{e\}\,, \\ e &= 0\,. \end{aligned}$$

Ce système se réduit en

$$\begin{aligned} a &= \{\{1\}, \{1, a\}\}\,, \\ b &= \{1\}\,, \\ c &= \{1, a\}\,, \\ d &= 1\,, \\ e &= 0\,. \end{aligned}$$

Il correspond donc à l'ensemble (non ordinaire) a "défini" par l'équation

$$a = \{\{1\}, \{1, a\}\}\,,$$

qui peut aussi s'écrire

$$a = (1, a)\,,$$

puisque le couple (x, y) est, par définition, l'ensemble $\{\{x\}, \{x, y\}\}$.

On peut remplacer l'axiome de fondation par un axiome exprimant que tout graphe accessible (bien fondé ou non) définit un ensemble, ou encore, admet une décoration unique.

Intuitivement, l'ensemble non ordinaire a correspond à $(1, (1, (1, \ldots)))$, l'objet infini solution de l'équation $a = (1, a)$. Ce type d'équation survient fréquemment en informatique théorique et sera étudié au chapitre 5.

L'introduction d'ensembles non ordinaires n'invalide naturellement pas le principe d'extensionnalité mais le rend quelque peu inopérant. Soit un ensemble non ordinaire b défini par $b = (1, b)$. Est-il égal à l'ensemble a qui vient d'être introduit ? Intuitivement, la réponse est oui : tous deux sont égaux à $(1, (1, (1, \ldots)))$. Le principe d'extensionnalité ne permet pourtant pas de l'affirmer : son application à l'identité des couples donne seulement la tautologie

$$a = b \equiv (1 = 1 \wedge a = b)\,.$$

Il est donc opportun de renforcer ce principe et d'admettre que deux ensembles sont égaux s'ils admettent une représentation (non étiquetée) commune.

On peut aussi remplacer l'isomorphisme de graphe par une relation d'équivalence plus faible, nommée *bisimulation*. En informatique, il est courant de représenter des processus par des graphes; deux processus seront jugés équivalents si les graphes les représentant sont bisimilaires. Ce terme vient lui-même de l'informatique; deux processus sont équivalents si chacun peut "simuler" le comportement de l'autre. La notion de bisimulation admet plusieurs variantes, engendrant ainsi plusieurs notions d'équivalence entre processus.

2.6. Cardinaux et axiome du choix

Le concept d'ordinal généralise celui de naturel en respectant l'idée intuitive de "comptage ordonné". On voudrait aussi généraliser la notion de "comptage simple". Nous avons vu que les nombres naturels sont modélisés par des ensembles; bien naturellement, le nombre naturel 5 (plus exactement, son modèle ensembliste) comporte cinq éléments. Les nombres naturels permettent tant le comptage simple que le comptage ordonné des ensembles finis. Le concept d'ordinal généralise adéquatement le concept de naturel en ce qui concerne le comptage ordonné d'ensemble infinis; un ordinal est un prototype d'ensemble bien ordonné. Nous examinons maintenant dans quelle mesure le comptage simple, ou dénombrement, peut se généraliser au cas infini.

2.6.1. *Equipotence*

Pour généraliser la notion de comptage simple, il faut d'abord en donner une description intrinsèque, valable tant dans le cas infini que dans le cas fini. Curieusement, la méthode de comptage parfois adoptée par les enfants présente ce caractère intrinsèque. Pour déterminer qu'un sac d'oranges contient cinq fruits, l'enfant "comptera sur ses doigts", c'est-à-dire qu'il associera, à chaque fruit, l'un de ses doigts, et en même temps, un nombre naturel (zéro exclu). Le dernier nombre cité sera le bon, c'est-à-dire le *cardinal* du sac d'oranges.

Plus généralement, on dira que deux ensembles ont *même cardinal*, ou sont *équipotents*, si leurs éléments peuvent être appariés, c'est-à-dire s'il existe une bijection entre les deux ensembles. La notation $a \simeq b$ exprime que les classes a et b sont équipotentes. Cette notation abrège la formule

$$\exists f\,[\mathit{fonc}(f) \wedge \mathit{dom}(f) = a \wedge \mathit{cod}(f) = b \wedge \forall x \forall y\,([x \in a \wedge y \in a \wedge f(x) = f(y)] \Rightarrow x = y)]\,.$$

Contrairement à ce que l'on pourrait croire, les notions de cardinal et d'ordinal divergent dans le cas infini. La notion de comptage simple étant moins riche que celle de comptage ordonné, on peut s'attendre à ce que des ordinaux distincts puissent avoir même cardinal.[74] On voit immédiatement que les ordinaux ω et ω', quoique distincts, ont même cardinal. En effet, la fonction qui à ω associe 0 et qui à tout naturel n associe le naturel $n+1$ est une bijection de domaine ω' et de codomaine ω. De même, l'ensemble des naturels et l'ensemble des entiers sont équipotents sans être isomorphe; aucun ordinal ne peut être isomorphe à l'ensemble des entiers, qui n'est pas bien ordonné (pour la relation d'ordre habituelle).

La relation $\simeq$ est une relation d'équivalence sur V. (L'équipotence concerne seulement les ensembles, et pas les classes propres; on peut aussi convenir que

[74] Comme on le verra plus loin, il est naturel d'identifier les cardinaux à des ordinaux particuliers.

toutes les classes propres sont équipotentes entre elles.)

On a vu que deux ordinaux équipotents ne sont pas nécessairement égaux, puisque ω et ω' sont équipotents. De manière un peu moins évidente, on a aussi $\omega \times 2 \simeq \omega$, $\omega \times \omega \simeq \omega$ et même $\omega^\omega \simeq \omega$. Par contre, ω et $\mathcal{P}(\omega)$ ne sont pas équipotents. Ceci est une conséquence du théorème de Cantor, que nous donnons ci-dessous.

Théorème 2.10. Si a est un ensemble, alors a et $\mathcal{P}(a)$ ne sont pas équipotents.
Preuve. Supposons par l'absurde que f soit une bijection de a sur $\mathcal{P}(a)$. On considère l'ensemble $b = \{x : x \in a \wedge x \notin f(x)\}$; cet ensemble est une partie de a et on a donc $f^{-1}(b) \in a$. Ceci donne lieu à une contradiction car on a $f^{-1}(b) \in b$ si et seulement si $f^{-1}(b) \notin f(f^{-1}(b))$, c'est-à-dire si et seulement si $f^{-1}(b) \notin b$.

Un ensemble est *fini* s'il est équipotent à un naturel; il suffit pour cela qu'il soit le codomaine d'une fonction dont le domaine est un naturel. Le cardinal d'un ensemble a fini est l'unique naturel avec lequel il est en bijection; c'est aussi le plus petit naturel n pour lequel il existe une fonction f de domaine n et de codomaine a. Un ensemble b est *infini* s'il n'est pas fini. Un ensemble infini est *dénombrable* s'il est équipotent à $\mathbf{N}$; il suffit pour cela que b soit le codomaine d'une fonction dont le domaine est $\mathbf{N}$.[75]

Si a est un ensemble, on note $\mathcal{P}_f(a)$ l'ensemble des parties finies de a. On a le théorème suivant.

Théorème 2.11. L'ensemble $\mathcal{P}_f(\mathbf{N})$ est infini dénombrable.
Preuve. Appelons *norme* d'un ensemble fini de naturels la somme de ces naturels. Il est clair que, pour tout naturel m, il existe un nombre fini de parties finies de $\mathbf{N}$ dont la norme est m. Il est donc possible d'énumérer l'ensemble des parties finies de $\mathbf{N}$ par ordre croissant des normes, ce qui montre que cet ensemble est équipotent à $\mathbf{N}$.

2.6.2. *Théorème de Schröder et Bernstein*

Il n'est pas toujours commode de construire des bijections; le théorème de Schröder et Bernstein permet d'établir l'équipotence de deux ensembles par la construction de deux injections. On introduit d'abord un lemme.

Lemme 2.12. Si les ensembles a, b et c sont tels que $a \subset b$, $b \subset c$ et $a \simeq c$, alors on a aussi $a \simeq b$ et $b \simeq c$.
Preuve. Le problème consiste à construire une bijection $g : a \mapsto b$, à partir d'une bijection $f : a \mapsto c$ (une bijection entre b et c s'obtient immédiatement

[75] On utilise souvent "dénombrable" au lieu de "fini ou dénombrable"; les deux terminologies ont leurs avantages. Les expressions "fini ou dénombrable" d'une part, et "infini dénombrable" d'autre part, permettent d'éviter toute ambiguïté.

par composition). Par rapport à f, un élément $x \in a$ peut se comporter de trois manières différentes, mutuellement exclusives :

- $f^{(n)}(x) \in a$ pour tout n;
- il existe un naturel n_b tel que $f^{(n)}(x) \in a$ pour tout $n < n_b$ et tel que $f^{(n_b)}(x) \in (b \setminus a)$;
- il existe un naturel n_c tel que $f^{(n)}(x) \in a$ pour tout $n < n_c$ et tel que $f^{(n_c)}(x) \in (c \setminus b)$.

(Rappelons que l'expression $f^{(n)}(x)$ abrège l'expression $f(f(\ldots(x)\ldots))$, où f est appliqué n fois; plus précisément, on définit par récurrence $f^{(0)}(x) = x$ et $f^{(n+1)}(x) = f(f^{(n)}(x))$.)
Soient X_b et X_c les sous-ensembles de a correspondant respectivement aux deux premiers et au troisième cas. La fonction $f|_{X_b}$ est une bijection de X_b sur $X_b \cup (b \setminus a)$ et la fonction $id|_{X_c}$ est une bijection de X_c sur X_c. Comme X_b et X_c partitionnent a tandis que $X_b \cup (b \setminus a)$ et X_c partitionnent b, la relation $g =_{def} (f|_{X_b} \cup id|_{X_c})$ est une fonction et une bijection de a sur b.
Remarque. Pour rendre la démonstration rigoureuse, il faut définir formellement X_b et X_c. On a $X_b = \bigcup\{z : [z \subset a \wedge \forall y\,(y \in z \Rightarrow f(y) \in (b \setminus a) \cup z)]\}$ et $X_c = a \setminus X_b$.

Théorème 2.13. Si a et b sont des ensembles et s'il existe une injection f de a dans b et une injection g de b dans a, alors a et b sont équipotents.
Preuve. La fonction f est bijective de a dans $f(a)$ et la fonction g est bijective de $f(a)$ dans $g(f(a))$. On a $g(f(a)) \subset g(b)$ et $g(b) \subset a$; comme il existe une bijection de $g(f(a))$ dans a (à savoir la fonction inverse de $(g \circ f)$, de domaine $g(f(a))$), il existe aussi une bijection h de $g(b)$ dans a, ce qui implique que $(h \circ g)^{-1}$ est une bijection de a dans b.

La relation d'égalité entre ensemble est l'intersection des relations d'inclusion et d'inclusion inverse entre ensembles. La relation d'équipotence donne lieu à une remarque analogue, vu le théorème de Schröder et Bernstein. On introduit la notation $a \lesssim b$ pour signifier qu'il existe une injection de a dans b. On a donc

$$(a \simeq b) \equiv [(a \lesssim b) \wedge (b \lesssim a)]\,.$$

Il est maintenant possible de "comparer les tailles" de deux ensembles a et b. Ces ensembles sont de même taille s'il existe une injection de l'un vers l'autre et réciproquement. Par contre, s'il existe une injection de a vers b sans qu'il existe une injection de b vers a, on dira que a est plus petit (ou de même taille) que b. Si $\#a$ désigne provisoirement le concept informel de "taille" de l'ensemble a, le théorème de Schröder et Bernstein implique que la relation entre les tailles induite par la relation $\lesssim$ est antisymétrique; elle est aussi réflexive et transitive, c'est donc une relation d'ordre. Intuitivement, cet ordre est total, mais on n'a pas formellement exclu le cas problématique où a et b seraient incomparables, c'est-à-dire le cas où il n'existerait d'injection ni dans un sens, ni dans l'autre. Un autre problème est de définir la notion même de cardinal, qui formaliserait

le concept de taille. Si on admet que tout ensemble est équipotent à un ordinal, on définira le cardinal d'un ensemble comme le plus petit ordinal équipotent à cet ensemble. En fait, les deux problèmes sont liés; avant d'approfondir ce point, nous illustrons brièvement la notion d'équipotence.

2.6.3. *Exemples*

Le théorème de Schröder et Bernstein est utilisé pour établir des résultats d'équipotence, tandis que le théorème de Cantor donne lieu à des résultats de non-équipotence.

Corollaire 2.14. L'ensemble des naturels, l'ensemble des entiers et l'ensemble des rationnels sont équipotents.
Preuve. Il est clair que l'ensemble des naturels et celui des entiers sont équipotents; la suite $(0, -1, 1, -2, 2, \ldots)$ est une bijection de $\mathbf{N}$ dans $\mathbf{Z}$.
On a déjà mentionné l'existence d'une injection de $\mathbf{Z}$ dans $\mathbf{Q}$. D'autre part, tout rationnel peut s'écrire sous la forme $\pm p/q$, où p et q sont des naturels premiers entre eux (le rationnel 0 reçoit la forme $+0/1$). La fonction f qui à $\pm p/q$ associe l'entier $\pm[(p+q)^2 + p]$ est une injection de $\mathbf{Q}$ dans $\mathbf{Z}$. Pour le voir, montrons comment p et q peuvent être déterminés à partir de $f(p/q)$. On cherche l'unique naturel n tel que $n^2 \leq f(p/q) < (n+1)^2$; on a alors $p+q = n$ et $p = f(p/q) - n^2$.

Corollaire 2.15. L'ensemble des naturels et l'ensemble des réels ne sont pas équipotents.
Idée de preuve. Vu le théorème de Cantor, il suffit de montrer que l'intervalle des réels compris entre 0 et 1 est équipotent à $\mathcal{P}(\mathbf{N})$. Pour cela, on caractérise un réel par son développement binaire illimité. Les suites infinies de 0 et de 1 sont visiblement en bijection avec les sous-ensembles de $\mathbf{N}$.
Remarque. Les intervalles réels $]0, 1[$ et $]-1, 1[$ sont équipotents, comme le montre la bijection $x \mapsto 2x - 1$; de même, l'intervalle $]-1, 1[$ est équipotent à l'ensemble $\mathbf{R}$, vu la bijection $x \mapsto x/(1 - |x|)$.

Les cardinaux des ensembles classiquement introduits en analyse sont évalués d'une manière analogue. Voici, sans preuve, deux résultats typiques.

Corollaire 2.16. L'ensemble des applications de $\mathbf{R}$ dans $\mathbf{R}$ est équipotent à l'ensemble $\mathcal{P}(\mathbf{R})$.

Corollaire 2.17. L'ensemble des applications continues de $\mathbf{R}$ dans $\mathbf{R}$ est équipotent à $\mathbf{R}$.

Nous interrompons ici l'étude des cardinaux pour introduire le dernier axiome de la théorie des ensembles. Cet axiome permettra (entre autres) d'assimiler les cardinaux à des ordinaux particuliers.

2.6.4. *Axiome du choix*

Nous avons vu au paragraphe 2.2.4 que la notion de couple et celle de produit cartésien binaire se généralisaient facilement. Rappelons qu'un élément de $\prod E$ est une fonction de choix sur E, c'est-à-dire une fonction f qui à tout élément $e \in E$ associe un élément $f(e) \in e$. La notion de fonction de choix généralise celle de couple: à tout $f \in \prod\{a,b\}$ correspond le couple $(f(a), f(b)) \in a \times b$, et à tout couple $(x,y) \in a \times b$ correspond la fonction de choix $\{(a,x),(b,y)\} \in \prod\{a,b\}$.

Nous avons vu aussi que l'ensemble $\prod E$ est vide si l'ensemble vide est un élément de E. La réciproque semble évidente et se prouve aisément si E est fini. On observe d'abord que, si a et b sont des ensembles, $a \times b$ n'est vide que si a est vide ou b est vide. On observe ensuite qu'il existe une bijection entre les ensembles $(\prod E) \times a$ et $\prod(E \cup \{a\})$, si E est un ensemble ne contenant pas l'ensemble a. Cela implique que les ensembles $(\prod E) \times a$ et $\prod(E \cup \{a\})$ sont tous deux vides ou tous deux non vides. On en déduit par récurrence sur la taille de l'ensemble fini E que $\prod E = \emptyset$ est équivalent à $\emptyset \in E$.

Remarque. Rappelons que $\prod \emptyset = \{\emptyset\}$ et que, pour tout ensemble a, $\prod\{a\}$ est assimilable à a. Si a est une classe propre, $\prod a$ est vide, car toute fonction de choix sur a est une classe propre.

Intuitivement, l'ensemble $\prod_{i \in I} E_i$ est un ensemble non vide dès qu'aucun des E_i n'est vide, même si l'ensemble I est infini. Un élément de ce produit s'obtient en "choisissant" un élément dans chaque E_i. Néanmoins, cette vérité intuitive implique, sauf quand I est fini, une infinité de choix et, en toute généralité, le processus de sélection n'est pas effectif. Ce point a conduit certains mathématiciens à rejeter cette vérité intuitive, ainsi que tous les théorèmes dans la preuve desquels elle est utilisée (souvent inconsciemment).

Remarque. L'acceptation ou le rejet de "l'axiome du choix" est une question en partie philosophique. Néanmoins, dans le cadre strict de la théorie des ensembles, on peut mettre en évidence trois points. Tout d'abord, l'axiome du choix se formalise aisément:

$$(2.10) \quad \forall x\,((ens(x) \wedge \emptyset \notin x) \Rightarrow \prod x \neq \emptyset)\,.$$

(ce schéma est noté AC). Ensuite, on peut montrer — c'est difficile — que l'axiome du choix n'est pas une conséquence des axiomes déjà introduits, mais que son adjonction n'induit pas d'inconsistance. Enfin, quoique l'axiome du choix soit intuitivement vrai, on peut démontrer qu'il est équivalent à d'autres énoncés dont la véracité n'est pas aussi intuitive.

2.6.5. *Diverses formulations de l'axiome du choix*

Ce paragraphe reprend certains énoncés classiques dont l'équivalence avec l'axiome du choix se démontre facilement.

Théorème 2.18. Les énoncés suivants sont tous équivalents.

1. Si E est un ensemble dont les éléments sont non vides et deux à deux disjoints, alors il existe un ensemble A tel que l'ensemble $A \cap e$ contienne un et un seul élément pour tout $e \in E$.
2. Si E est un ensemble, alors il existe une fonction f qui associe à toute partie non vide de E un élément de cette partie. On a donc $f(A) \in A$ pour tout $A \in \mathcal{P}(E) \setminus \{\emptyset\}$.
3. Si les éléments de l'ensemble E sont non vides, alors $\prod(E)$ est non vide.
4. Si R est une relation dont le domaine est l'ensemble A,
   alors il existe une fonction f de domaine A telle que $f \subset R$.
5. Soit E un ensemble ordonné et soit $\mathcal{C}(E)$ l'ensemble des chaînes de E. L'ensemble $\mathcal{C}(E)$, ordonné par inclusion, admet un élément maximal.
6. Si E est un ensemble ordonné dont toute chaîne admet une meilleure borne supérieure dans E, alors l'ensemble E admet un élément maximal.
7. Tout ensemble E peut être muni d'un bon ordre.

Remarque. Les quatre premiers énoncés sont visiblement équivalents, et habituellement confondus sous le nom d'axiome du choix. Les trois premiers sont attribués à Zermelo, et le quatrième est attribué à Baer. Les trois derniers énoncés sont intuitivement peu évidents. (Par exemple, on ne voit pas de moyen simple pour bien ordonner l'ensemble des nombres réels.) L'énoncé 5 est le "principe de Hausdorff" et l'énoncé 6 est le "lemme de Zorn". On énonce parfois le principe de Hausdorff sous une forme apparemment plus forte, attribuée à Kuratowski : dans un ensemble ordonné, toute chaîne est incluse dans une chaîne maximale. L'équivalence de cet énoncé avec les précédents se démontre facilement.

Preuve. On montre d'abord l'équivalence des énoncés 5 et 6.
Soit d'abord un ensemble ordonné E dont toute chaîne admet une meilleure borne supérieure dans E. Le principe de Hausdorff garantit l'existence d'une chaîne maximale C. La meilleure borne supérieure de C, notée c, appartient à E (et à C); c'est un élément maximal de E, car s'il existait un d supérieur à c, la chaîne C se prolongerait en la chaîne $C \cup \{d\}$ et ne serait pas maximale. Ceci établit le lemme de Zorn.
D'autre part, soit E un ensemble ordonné. L'ensemble $\mathcal{C}(E)$ est ordonné par inclusion et la meilleure borne supérieure d'un élément $C \in \mathcal{C}(\mathcal{C}(E))$ est simplement la chaîne $\bigcup C$; le lemme de Zorn, appliqué à l'ensemble $\mathcal{C}(E)$, garantit l'existence d'un élément maximal $C \in \mathcal{C}(E)$. Cet élément est une chaîne maximale, ce qui établit le principe de Hausdorff.
Prouvons ensuite l'équivalence de l'axiome du choix et de l'énoncé 7.
D'une part, si l'ensemble E est bien ordonné, la fonction qui à chaque partie non vide A de E associe le minimum de A valide l'axiome du choix (énoncé 2).

D'autre part, pour munir un ensemble E d'un bon ordre, il est suffisant de le mettre en bijection avec un ordinal (§ 2.5.3). Si f associe à toute partie non vide A de E un élément $f(A) \in A$, on considère la fonction h de Ω dans $\mathcal{P}(E)$ définie comme suit :

- $h(0) =_{def} E$;
- $h(r') =_{def} h(r) \setminus \{f(h(r))\}$, pour tout ordinal r ;
- $h(\ell) =_{def} \bigcap\{h(r) : r \in \ell\}$, pour tout ordinal limite ℓ.

On pose aussi $f(\emptyset) = V$, d'où $\{f(\emptyset)\} = \emptyset$.
Le théorème de von Neumann garantit l'existence et l'unicité de la fonction h. On a $h(r) \subset E$ pour tout r et, si $s \in r$, alors $h(s) \subset h(r)$; cette inclusion est stricte sauf si $h(s)$ est vide. En effet, si $h(s)$ n'est pas vide, $h(r) \setminus h(s)$ contient $f(h(s))$. La classe $B =_{def} \{r : r \in \Omega \wedge h(r) = \emptyset\}$ est non vide, sinon h serait une injection de la classe propre Ω dans l'ensemble $\mathcal{P}(E)$, ce qui est impossible. Soit donc $\lambda = \bigcap B$ le minimum de B. La composée $g = f \circ h$, restreinte au domaine λ, est injective car, si $s \in r$ et $r \in \lambda$, alors $f(h(s)) \notin h(r)$, d'où $f(h(r)) \neq f(h(s))$. La fonction g est aussi surjective: si $r \in \lambda$, on a $x = g(r)$ si $x \in h(r) \setminus h(r')$, c'est-à-dire si $r = \bigcup\{s : s \in \Omega \wedge x \in h(s)\}$. La fonction g est donc une bijection entre λ et E, et on peut poser $x < y$ dans E si $g^{-1}(x) \in g^{-1}(y)$ dans λ, ce qui donne un bon ordre sur E.[76]
On termine la preuve en déduisant le lemme de Zorn de l'axiome du choix. Soit E un ensemble ordonné dont toute chaîne admet une meilleure borne supérieure, et soient x un élément de E et f une fonction qui à chaque partie non vide A de E associe un élément de A. On définit par induction ordinale une fonction h de Ω dans $\mathcal{P}(E)$ comme suit :

- $h(0) =_{def} \{y : y \in E \wedge x < y\}$;
- $h(r') =_{def} \emptyset$, pour tout ordinal r tel que $h(r) = \emptyset$;
- $h(r') =_{def} \{y : y \in E \wedge f(h(r)) < y\}$, pour tout r tel que $h(r) \neq \emptyset$;
- $h(\ell) =_{def} \bigcap\{h(r) : r \in \ell\}$, pour tout ordinal limite ℓ.

Soit λ le plus petit ordinal tel que $h(\lambda) = \emptyset$. Si $\lambda = 0$, x est maximal dans E; sinon, l'ensemble $\{f(h(r)) : r \in \lambda\}$ est non vide ($f(h(0))$ en est membre) et est une chaîne (on a $f(h(r_1)) < f(h(r_2))$ si $r_1 < r_2$). La meilleure borne supérieure de cet ensemble existe et est un élément maximal de E.

---

[76] La technique utilisée ici permet aussi de déterminer l'unique ordinal λ isomorphe à un ensemble E muni d'un bon ordre donné a priori. Il suffit d'utiliser comme fonction de choix f celle qui à toute partie non vide A de E associe le minimum de A. Le principe de Hausdorff permet également la construction d'un bon ordre sur un ensemble E. Ce principe, appliqué à l'ensemble $\mathcal{P}(E)$, ordonné par inclusion, garantit l'existence d'une chaîne maximale A. Cette chaîne, composée de parties de E, est bien ordonnée : si B est un sous-ensemble non vide de A, le minimum de B est $\bigcap B$. On obtient un bon ordre sur E en posant $x < y =_{def} \exists u\,[u \in A \wedge x \in u \wedge y \notin u]$.

Les quelques énoncés dont nous venons de prouver l'équivalence avec l'axiome du choix proviennent tous de la théorie des ensembles. En fait, diverses propositions importantes en algèbre, en analyse et en topologie sont vraies si et seulement si l'axiome du choix est vrai; on comprend donc que l'étude de cet axiome se poursuive sans interruption depuis un siècle.
Rappelons aussi que l'axiome du choix est indépendant des axiomes antérieurs (il n'est donc pas question de prouver l'axiome du choix) et que son adjonction aux axiomes antérieurs n'altère pas la consistance de la théorie (il n'est donc pas question non plus de réfuter l'axiome du choix).
Il n'est pas possible de prouver la consistance de la théorie des ensembles dans le cadre de la théorie elle-même, pas plus qu'il n'est possible de prouver la consistance de l'arithmétique dans le cadre de l'arithmétique. Par contre, comme nous l'avons vu précédemment, les postulats de Peano deviennent des théorèmes de la théorie des ensembles; si cette dernière est consistante, l'arithmétique de Peano l'est aussi.

2.6.6. *Application à la théorie des cardinaux*

Nous avons vu au paragraphe 2.6.2 que l'on pouvait s'attendre à ce que les cardinaux forment une sous-classe des ordinaux. Cette vérité intuitive est une conséquence immédiate de l'axiome du choix (énoncé 7). En effet, puisque tout ensemble peut être bien ordonné, tout ensemble est équipotent à un ordinal. Si E est un ensemble, la classe des ordinaux équipotents à E est non vide et admet un minimum. On définira donc le cardinal $|E|$ de E comme le plus petit ordinal équipotent à E.

On observe aussi que la relation $\lesssim$ vérifie la formule

$$\forall a \forall b\, [(a \lesssim b) \vee (b \lesssim a)]\,;$$

cette relation est donc totale. Cette propriété de totalité est en fait équivalente à l'axiome du choix. On a vu qu'elle en était une conséquence directe et, d'autre part, si tous les ensembles sont comparables, tout ensemble est comparable à tout ordinal, ce qui permet la définition "ordinale" des cardinaux, et implique que tout ensemble admet un bon ordre.

Remarque. On voit immédiatement qu'une classe a est propre si et seulement si $r \lesssim a$ pour tout ordinal r, ou encore si $r \not\approx a$ pour tout ordinal r. Ceci justifie la convention déjà évoquée: deux classes propres sont toujours équipotentes.

Les cardinaux finis s'identifient aux ordinaux finis, mais les cardinaux infinis forment une sous-classe (propre) de la classe des ordinaux infinis. Le plus petit cardinal infini est ω, et il semble difficile d'en construire un plus grand. En effet, tous les ordinaux obtenus par addition, multiplication et exponentiation d'ordinaux dénombrables restent dénombrables. Néanmoins, une conséquence immédiate du théorème de Cantor est l'existence d'une infinité de cardinaux infinis, respectivement équipotents à $\mathbf{N}$, $\mathcal{P}(\mathbf{N})$, $\mathcal{P}(\mathcal{P}(\mathbf{N}))$, etc.

On décrit plus précisément la classe des cardinaux infinis au moyen d'une fonction spéciale, notée $\aleph$ (lire "aleph", première lettre de l'alphabet hébreu) et définie par induction ordinale sur la classe Ω. On écrit souvent $\aleph_\sigma$ plutôt que $\aleph(\sigma)$. On pose $\aleph_0 = \omega$. Pour tout ordinal successeur r', $\aleph_{r'}$ est le plus petit cardinal plus grand que $\aleph_r$. Pour tout ordinal limite ℓ, $\aleph_\ell$ est $\bigcup\{\aleph_s : s \in \ell\}$.

Le théorème suivant décrit la structure de la classe des cardinaux infinis.

Théorème 2.19. Les cardinaux infinis forment une classe propre, qui est le codomaine de l'isomorphisme $\aleph$.

Preuve. On vérifie d'abord que la définition de la fonction $\aleph$ est licite. D'une part, si le cardinal $\aleph_r$ est correctement défini, la classe des cardinaux plus grands que $\aleph_r$ n'est pas vide (elle comporte en particulier le cardinal de l'ensemble $\mathcal{P}(\aleph_r)$); cette classe se compose d'ordinaux et admet donc un minimum. D'autre part, si $\aleph_s$ est correctement défini pour tout ordinal $s \in \ell$, l'ensemble $\aleph_\ell$ existe et est un ordinal. Soit α le plus petit cardinal plus grand que tous les $\aleph_s$ et soit β le cardinal de $\aleph_\ell$. Il faut prouver $\alpha = \aleph_\ell = \beta$. D'abord, comme $\aleph_\ell$ est le plus petit *ordinal* plus grand que tous les $\aleph_s$, et que tout cardinal est un ordinal, on a $\aleph_\ell \leq \alpha$. D'autre part, le cardinal d'un ordinal est au plus égal à cet ordinal, d'où $\beta \leq \aleph_\ell$. Enfin, β est plus grand que tous les $\aleph_s$, sinon il s'identifierait à un $\aleph_s$, et serait plus petit que le cardinal suivant $\aleph_{s'}$; on a donc $\alpha \leq \beta$, d'où le théorème.

Ce théorème semble indiquer que la classe des cardinaux infinis est un objet simple, dont les propriétés se déduisent facilement de celles de la classe des ordinaux. En fait, il n'en est rien, et la théorie des cardinaux suscite plus de problèmes qu'elle n'en résout. Tout d'abord, le théorème de Cantor permet d'écrire $\mathcal{P}(\aleph_0) \simeq \aleph_\alpha$, mais la valeur de α n'est pas connue. L'hypothèse la plus "naturelle" est $\mathcal{P}(\aleph_0) \simeq \aleph_1$; elle est due à Cantor et on la nomme souvent "hypothèse du continu", en référence à sa forme équivalente $\mathbf{R} \simeq \aleph_1$. Le statut de cette formule est analogue à celui de l'axiome du choix: il n'est pas possible de la prouver ou de la réfuter dans le cadre de la théorie. Il existe donc une grande variété d'univers ensemblistes (du moins si la théorie est consistante!) qui se différencient par la valeur de vérité qu'ils attribuent à l'axiome du choix, l'hypothèse du continu, et bien d'autres énoncés. Les recherches actuelles en théorie des ensembles visent à éclaircir la signification profonde de ces énoncés indépendants de la théorie de base, ainsi que les interactions entre eux. Plus précisément, on cherche des axiomes supplémentaires dont ces énoncés (ou leur négation) soient conséquences logiques. Diverses tentatives infructueuses ont enrichi le répertoire d'énoncés non décidés de la théorie. Par exemple, on a visiblement $\sigma \leq \aleph_\sigma$ pour tout ordinal σ, et on s'attend à ce que l'inégalité soit stricte. A nouveau, ceci n'a pu être prouvé et, au contraire, il est apparu que l'existence d'un cardinal $\gamma = \aleph_\gamma$ était envisageable et avait des conséquences intéressantes. Bien que bon nombre de découvertes surprenantes de ce type aient été faites, la question n'est pas résolue à l'heure actuelle.

2.6.7. *Application aux définitions récursives*

Si (Γ, R) est un bon ordre, la théorie de von Neumann permet de définir récursivement une fonction f de domaine Γ au moyen d'un schéma du type $f(a) = e_a$, pour tout $a \in \Gamma$, à condition que l'expression e_a n'évoque des formes de type $f(b)$ que pour des éléments b strictement inférieurs à a (pour l'ordre R). Nous avons observé que la technique restait intuitivement valide si la relation R était bien fondée sans être totale.

Considérons un exemple simple. La classe Γ comporte les arbres binaires finis dont chaque nœud est étiqueté par (identifié à) un nombre naturel. La relation d'ordre bien fondé strict R associée à A est la relation structurelle: aRb signifie que a est un sous-arbre de b. Cette relation n'est pas totale. Les éléments minimaux de A sont tous les arbres réduits à un nœud unique; ils forment un ensemble infini dénombrable. Un élément non minimal $x \in \Gamma$ est une structure $[r, y, z]$ où r est la racine (un naturel) et où y et z sont les sous-arbres de gauche et de droite (éléments arbitraires de Γ), respectivement. Il est intuitivement valide de définir sur le domaine Γ la fonction σ par les énoncés

$$\sigma(r) = r\,, \text{ si } r \text{ est minimal, c'est-à-dire si } r \in \mathbf{N},$$
$$\sigma(r) = n + \sigma(x) + \sigma(y)\,, \text{ si } r = [n, x, y].$$

On peut valider la technique sans refaire la théorie de von Neumann en observant que la relation bien fondée R est incluse dans une relation de bon ordre S mais, même dans l'exemple élémentaire qui vient d'être introduit, la construction d'un bon ordre compatible avec R (incluant R) est fastidieuse. Il est préférable de démontrer, une fois pour toutes, le théorème suivant.

Théorème 2.20. Si R est une relation d'ordre strict bien fondé sur l'ensemble Γ, alors il existe un bon ordre strict S sur Γ tel que $R \subset S$.

Remarque. Il est impossible de prouver ce théorème sans l'axiome du choix, puisque ce dernier en est une conséquence immédiate (on le voit en choisissant pour la version stricte de R la relation vide). Prouver ce théorème revient donc à montrer son équivalence avec l'axiome du choix.

Preuve. Soit $\mathcal{P}_0(\Gamma)$ l'ensemble des parties non vides de Γ. Soit f, la fonction qui, à tout $\gamma \in \mathcal{P}_0(\Gamma)$ associe l'ensemble des éléments R-minimaux de γ. La relation R étant bien fondée, $f(\gamma)$ n'est jamais vide. Le codomaine Δ de la fonction f est donc un ensemble d'ensembles non vides et, par l'axiome du choix, il admet une fonction de choix g. La fonction $m = g \circ f$ associe donc à toute partie non vide $\gamma \subset \Gamma$ un élément minimal de cette partie; on pose aussi $m(\emptyset) = \perp$, où $\perp$ désigne une valeur quelconque n'appartenant pas à Γ. On utilise alors le schéma de von Neumann pour définir une fonction h de domaine Ω et de codomaine inclus dans $\Gamma \cup \{\perp\}$; on pose

$$h(\alpha) = m(\Gamma \setminus cod(h|_\alpha))\,,$$

pour tout ordinal α. Soit alors β le plus petit ordinal tel que $h(\beta) = \perp$, c'est-à-dire $cod(h|_\beta) = \Gamma$. On démontre facilement que cet ordinal existe et que $h|_\beta$

est une bijection de domaine β et de codomaine Γ. Cette bijection induit le bon ordre demandé. Pour tous $\alpha, \alpha' \in \beta$, on a $(h(\alpha), h(\alpha')) \in S$ si et seulement si $\alpha \in \alpha'$; en outre, dans Γ, si xRy, alors xSy, c'est-à-dire $h^{-1}(x) \in h^{-1}(y)$.

2.6.8. *Statut de l'axiome du choix*

On utilise l'axiome du choix, comme n'importe quel autre axiome, pour déduire des conclusions jugées intéressantes. Néanmoins, cet axiome a suscité — et suscite encore — beaucoup de controverses. On estime souvent qu'il n'est pas "aussi intuitivement vrai" que les autres axiomes de la théorie, ce qui ne l'a pas empêché d'être utilisé intensivement depuis Cantor, souvent de manière implicite. Avant même que la théorie axiomatique des ensembles n'ait atteint une certaine maturité, les mathématiciens se sont attachés à débusquer les usages occultes de l'axiome du choix. Dans ce chapitre, nous avons essayé de ne pas l'utiliser avant son introduction ... ce qui ne signifie pas que nous ayons réussi, tant les déconvenues dans ce domaine ont été nombreuses.

Du point de vue du théoricien ensembliste formaliste, l'axiome du choix est intéressant parce qu'il est indépendant des autres axiomes, tout en n'induisant pas d'inconsistance. Dans ce cadre, il est indiqué de préciser si un énoncé démontré grâce à l'axiome du choix pourrait encore être démontré sans cet axiome. Plus généralement, le théoricien s'intéressera à des variantes de l'axiome du choix, voire à des énoncés contredisant l'axiome du choix tout en restant compatibles avec les autres axiomes de la théorie. En outre, d'autres énoncés peuvent aussi être ajoutés à la théorie sans induire d'inconsistance; ces énoncés, dont le plus connu est l'hypothèse du continu, déjà mentionnée, font également l'objet d'études théoriques. En fait, ces énoncés et les systèmes axiomatiques auxquels ils donnent lieu constituent l'objet même de la recherche des mathématiciens formalistes.

Certaines théories, comme celle des cardinaux, restent embryonnaires si l'axiome du choix n'est pas admis, tandis que d'autres sont indépendantes de cet axiome. Parfois, une version affaiblie de l'axiome suffit; c'est le cas en analyse élémentaire, où on admet généralement que le produit d'un ensemble *dénombrable* d'ensembles non vides est non vide. D'un point de vue technique, le point n'est plus de décider si l'axiome du choix doit être admis ou non, mais plutôt si tel énoncé ou telle théorie en dépend.

D'un point de vue plus pragmatique, on peut décider d'accepter inconditionnellement l'axiome du choix, dans le but de simplifier certaines démonstrations. Par exemple, si on accepte l'axiome du choix, le théorème de Schröder et Bernstein devient évident, puisque tout ensemble devient équipotent à un ordinal. Nous avons utilisé l'axiome du choix pour faire l'économie d'une extension supplémentaire de la théorie de von Neumann. On peut aussi l'utiliser pour définir la notion d'ensemble fini sans référence aux naturels: un ensemble a est infini s'il existe un ensemble b strictement inclus dans a et équipotent à a.

Dans le cadre de l'axiome du choix, il suffit de considérer le cas des ordinaux. On voit, d'une part, que la propriété est fausse pour les naturels, et d'autre part que ω est équipotent à $\omega \setminus \{0\}$, d'où on déduit immédiatement que α est équipotent à $\alpha \setminus \{0\}$, pour tout ordinal α contenant ω.

Il n'est pas interdit de continuer à spéculer sur le caractère "intuitivement vrai" de l'axiome du choix. Une telle démarche peut contribuer à mieux appréhender le "modèle standard" de la théorie des ensembles (s'il existe). Ce type de réflexion est justifié par le précédent de la théorie axiomatique des nombres. Celle-ci admet une infinité de modèles, dont le modèle arithmétique standard, qui peut se définir rigoureusement dans certains formalismes. On peut néanmoins penser que les progrès en ce domaine sont conditionnés par les progrès à faire dans le cadre strict de la théorie axiomatique.

2.7. Théorie des ensembles et théorie des types

2.7.1. *Approches alternatives au problème des types*

Dans ce chapitre, nous avons étudié la notion d'ensemble et mis en évidence les divers problèmes qu'elle suscite. Une longue introduction a permis de justifier la présentation de la théorie axiomatique des ensembles; faute de motivation, cette théorie paraît souvent artificielle au premier abord. Dans le même esprit, avant d'introduire les axiomes et théorèmes principaux de la théorie, il était utile de résumer les problèmes qui ont conduit à l'abandon relatif de la théorie naïve. Nous complétons maintenant le tableau en évoquant brièvement des solutions alternatives, qui permettent en particulier d'éviter le paradoxe de Russell.

Rappelons d'abord que, pour le logicien classique, le concept d'ensemble n'appartient pas à la logique. Dans une théorie du premier ordre, un domaine d'interprétation est un ensemble d'objets, et les variables logiques désignent ces objets, et non l'ensemble (ou un sous-ensemble) de ces objets. En logique classique, le mot "ensemble" appartient au métalangage. La création d'une théorie des ensembles du premier ordre présente donc d'emblée une difficulté : on doit distinguer entre "ensemble", objet de la théorie et élément du domaine d'interprétation (appelé souvent "univers") et "méta-ensemble", collection de tout ou partie des éléments de l'univers.

A première vue, le choix d'une terminologie adaptée permet d'éviter cet écueil. Toutefois, même si on se limite aux ensembles-objets, c'est-à-dire aux éléments de l'univers, le problème resurgit, de manière amplifiée. Nous avons signalé, au paragraphe 2.2, que les ensembles forment une hiérarchie de types. En géométrie plane, par exemple, il est commode de considérer que les points sont des éléments fondamentaux, de type 0; les droites et le plan sont des ensembles de type 1, tandis que l'ensemble des droites du plan est de type 2. Plus généralement, dans un univers, c'est-à-dire un domaine d'interprétation dont les éléments sont des ensembles-objets (ou ensembles formels),

- un ensemble formel u est de type 0 s'il n'existe pas d'ensemble formel v tel que $v \in u$;
- un ensemble w sera de type n s'il contient un élément de type $n-1$ et si tous ses éléments sont au plus de type $n-1$.

Cette notion de type, quoique simple en apparence, est en fait fort délicate. D'une part, il apparaît difficile d'éviter les "types infinis"; d'autre part, introduire des objets de type 0, dits *éléments fondamentaux*,[77] est déjà critique. D'après le principe d'extensionnalité, en effet, il ne peut exister qu'un seul ensemble formel de type 0, à savoir l'ensemble vide, noté $\emptyset$.

Nous avons vu au paragraphe 2.3 comment des objets apparemment fondamentaux sont assez naturellement et très efficacement modélisés par des ensembles formels de type non nul. Par exemple, les points du plan euclidien sont assimilés aux couples de nombres réels; les nombres réels sont construits à partir des naturels (via les entiers relatifs et les rationnels), tandis que les naturels sont inductivement construits à partir de l'ensemble vide.

Cette approche est très raisonnable ... à condition de ne pas devoir trop se soucier du type des objets manipulés. Dans le contexte de la théorie élémentaire des nombres et des fonctions, la détermination du type de la fonction exponentielle $x \longrightarrow e^x$, par exemple, est un problème non trivial, quoique dépourvu d'intérêt. Les paradoxes, et en particulier le paradoxe de Russell, montrent néanmoins que la définition de "collections" de type indéterminé peut être catastrophique. La solution classiquement adoptée consiste à n'admettre comme ensemble formel que les collections obtenues au départ de l'ensemble vide, à partir de règles constructives strictes. Cela donne lieu à la théorie dite de Zermelo et Fraenkel, ou théorie *ZF*, dont nous avons présenté une variante dans ce chapitre.

La concision est sans doute la qualité primordiale de l'approche *ZF*.[78] Les théorèmes ont généralement une forme courte et élegante. Cette concision a pourtant son prix, et l'approche *ZF* ne fait pas l'unanimité. La nature implicite des types suscite d'abord une objection de nature philosophique : si la notion de type est si importante, pourquoi vouloir la dissimuler ? Des considérations plus pragmatiques étayent cette objection. Par exemple, faute de typage explicite, l'adepte de la théorie *ZF* est contraint d'accepter des égalités pathologiques telles que $(0,0) = \{1\}$ (§ 2.3.6). Plus fondamentalement, le relatif laxisme avec lequel la notion de type est traitée dans le cadre de la théorie *ZF* rend celle-ci assez peu discriminante : elle admet des modèles extrêmement variés, aux

[77] On utilise souvent le terme allemand *Urelemente*.

[78] Il peut être préférable de parler d'approche *ZF* plutôt que de théorie, parce que les nombreuses variantes de la théorie *ZF* (dues à Morse, Gödel, Bernays, etc.) jouissent des principales qualités de celle-ci (en particulier la concision), reposent sur des bases fort voisines et ne sont pas sans mérites propres. L'étude des différentes variantes sort du cadre de ce livre.

propriétés fort différentes, alors que l'univers des ensembles semble une réalité quasi aussi définie que la collection des nombres naturels.

Le formaliste réfutera ces objections en présentant la théorie ZF comme une théorie du premier ordre dont la signature comporte le symbole d'appartenance et éventuellement la constante V. Un modèle sera simplement un graphe, dont les nœuds sont les classes et dont les arcs représentent la relation d'appartenance. Les axiomes de la théorie ZF imposent certaines contraintes sur cette relation. La diversité de ces graphes-modèles sera vue comme un avantage, montrant la richesse de la théorie. Cette position extrême implique de renoncer à l'interprétation intuitive du symbole d'appartenance, ce qui est dommage. Même dans les domaines les plus abstraits des mathématiques, l'intuition reste un guide précieux. Nous allons donc présenter très brièvement des formalismes impliquant l'usage explicite des types.

Une solution partielle très simple, du moins en principe, est fournie par la logique dite du second ordre. Dans cette logique, on distingue les variables de type 0, représentant des éléments d'un domaine D, et les variables de type 1, représentant des fonctions ou des relations sur D, c'est-à-dire des sous-ensembles adéquats de D^n. La puissance d'expression est fortement augmentée, même si les objets de type élevé restent inaccessibles. Malheureusement, il n'est pas possible d'obtenir une axiomatisation complète des théories du second ordre, ni d'énumérer les théorèmes.

Dans certains contextes, ces lacunes sont acceptables et rien n'empêche l'utilisation de logiques d'ordre plus élevé. Une logique d'ordre trois comportera des constantes interprétées comme des objets de type 2, et une logique d'ordre quatre comportera en outre des variables (quantifiables) représentant des objets de type 2 (fonctions de fonctions, par exemple). Il faut cependant remarquer que même des problèmes plus simples que celui de la validité ou de la déductibilité deviennent fort complexes et même indécidables pour des logiques d'ordre élevé. Un exemple classique est celui de l'unification (§I.6.4). Rappelons ici que l'unification, jointe à la règle de résolution, est la clef de la démonstration automatique. Elle permet de rapprocher des clauses telles que

$$E(f(3),z)\,,\ (\neg E(f(x),v)\ \vee\ E(f(x+1),(x+1)\,.\,v))\,.$$

L'unification proprement dite fournit les clauses dérivées

$$E(f(3),z)\,,\ (\neg E(f(3),z)\ \vee\ E(f(4),4\,.\,z))\,,$$

puis la résolution fournit[79]

$$E(f(4),4\,.\,z)\,.$$

La découverte de la substitution unifiante, appelée *unificateur principal*, est un problème facile dans le cadre de la logique du premier ordre; ce n'est plus le cas si on considère des logiques d'ordre plus élevé.

[79] Ceci peut s'interpréter comme suit : si la factorielle de 3 vaut z, alors la factorielle de 4 vaut $4\,.\,z$.

2.7.2. *Introduction à la théorie formelle des types*

En logique du second ordre, on distingue constantes et variables, ainsi qu'objets de type 0 et objets de type 1. L'usage de quatre alphabets suffit donc à identifier les quatre catégories de symboles. Pour des logiques d'ordre plus élevé, il devient indispensable de noter explicitement les types. On se donne deux types de base, notés i (individus) et v (valeurs de vérité). La règle inductive est la suivante: $(\tau_1 \dots \tau_n)$ est le type des relations à n arguments de types respectifs $\tau_1, \dots, \tau_n$. On n'introduit pas les fonctions, car une fonction à n arguments peut toujours être remplacée par une relation à $n+1$ arguments. Cette notion de type est plus précise que celle introduite au paragraphe 2.7.1. Elle n'implique plus l'assimilation des relations à des ensembles, et permet l'usage sans restriction du schéma de compréhension. Si τ est un type et si $A(x_\tau)$ est une formule dont la seule variable libre est x_τ, on a

$$\exists\, r_{(\tau)}\, \forall\, x_\tau\, [r_{(\tau)}(x_\tau) \equiv A(x_\tau)]\,,$$

ce qui signifie que les objets (de type τ) vérifiant $A(x_\tau)$ admettent une "fonction caractéristique", c'est-à-dire forment un ensemble.[80] C'est l'usage de types explicites qui empêche l'apparition des paradoxes; en particulier, il n'est plus possible d'instancier $A(x_\tau)$ en $\neg x_\tau(x_\tau)$ (c'est-à-dire en $\neg x_\tau \in x_\tau$), car cette instantiation est syntaxiquement incorrecte. Le paradoxe de Russell est donc éliminé, de manière plus directe que dans l'approche *ZF*.

Le schéma d'extensionnalité devient

$$\forall\, x_\tau\, \forall\, y_\tau\, [x_\tau = y_\tau \;\equiv\; \forall\, z_{(\tau)}\, (z_{(\tau)}(x_\tau) \equiv z_{(\tau)}(y_\tau))]\,.$$

Si $\tau = (\tau_1 \tau_2)$, on peut aussi écrire

$$\forall\, x_{(\tau_1\tau_2)}\, \forall\, y_{(\tau_1\tau_2)}$$
$$[x_\tau = y_\tau \;\equiv\; \forall\, z1_{\tau_1}\, \forall\, z2_{\tau_2}\, (x_{(\tau_1\tau_2)}(z1_{\tau_1}, z2_{\tau_2}) \equiv y_{(\tau_1\tau_2)}(z1_{\tau_1}, z2_{\tau_2}))]\,.$$

On voit que les schémas de compréhension et d'extensionnalité sont de nature purement logique, un fait important que masque l'approche *ZF*.

L'écriture explicite des types conduit à un alourdissement considérable des formules. Toutefois, cet alourdissement peut être limité, et va de pair avec une augmentation considérable du pouvoir d'expression. En effet, il est maintenant possible d'évoquer *directement* une relation, plutôt que d'en faire un ensemble de couples, eux-mêmes modélisés par des paires d'ensembles. L'approche explicite a donc évolué de manière à optimiser l'expressivité et la concision du formalisme.

Il est apparu notamment que la notion de fonction était supérieure à celle de relation comme concept de base de la typification. Etant donné deux types α et β, on note $(\alpha \to \beta)$ ou simplement $(\alpha\beta)$ le type des fonctions appliquant

[80] Une relation à un argument représente bien un ensemble, comme on le verrait en écrivant $x_\tau \in r_{(\tau)}$ au lieu de $r_{(\tau)}(x_\tau)$.

des objets de type α sur des objets de type β.[81] Une relation à un argument de type α (modélisant un ensemble) sera une fonction de type (αv). Pour réduire les fonctions à plusieurs arguments à des fonctions à un seul argument, on utilise la technique de Curry (§ 2.3.4.7), basée sur l'isomorphisme existant entre les ensembles fonctionnels $((\alpha \times \beta) \rightarrow \gamma)$ et $(\alpha \rightarrow (\beta \rightarrow \gamma))$. Le premier ensemble comporte des fonctions à deux arguments de types respectifs α et β, appliqués sur des valeurs de type γ. Le second ensemble se compose de fonctions appliquant *un* argument de type α sur une valeur (fonctionnelle) de type $(\beta \rightarrow \gamma)$. Un prédicat à deux arguments de types respectifs α et β sera modélisé par une fonction de type $(\alpha \rightarrow (\beta \rightarrow v))$, ou simplement $(\alpha(\beta v))$.

Reconsidérons, avec les notations fonctionnelles, l'instance du schéma de compréhension introduite plus haut :

$$\exists\, r_{(\tau v)}\, \forall\, x_\tau\, [r_{(\tau v)}(x_\tau) \equiv A_v(x_\tau)]\,.$$

Cette instance reste valide si on remplace le type v par un type quelconque σ; on obtient

$$\exists\, r_{(\tau \sigma)}\, \forall\, x_\tau\, [r_{(\tau \sigma)}(x_\tau) \equiv A_\sigma(x_\tau)]\,.$$

Etant donné une expression quelconque $A_\sigma(x_\tau)$ prenant des valeurs de type σ et comportant x_τ comme variable libre, la formule affirme l'existence d'une fonction $r_{(\tau \sigma)}$ (nécessairement unique) dont la valeur coïncide avec celle de l'expression, pour n'importe quelle instance licite de x_τ. Cet énoncé semble banal, mais exprime l'essence même de la notion de fonction. Cette notion permet, précisément, d'abstraire en un seul objet une "loi de correspondance" entre objets (laquelle loi serait représentée par un ensemble de couples dans l'approche *ZF*). Il suffira donc d'introduire une notation pour la fonction en question pour obtenir une "théorie axiomatique des fonctions" au moins aussi puissante qu'une théorie axiomatique des ensembles. Conformément à l'usage, on désigne par $\lambda x_\tau(A_\sigma(x_\tau))$ la fonction qui, à chaque valeur licite de x_τ, associe la valeur correspondante de $A_\sigma(x_\tau)$. Ce mécanisme d'abstraction est le point de départ du "calcul lambda", dont l'étude sortirait du cadre de ce livre (voir par exemple Andrews 1965).

[81] Certains auteurs écrivent $(\beta\alpha)$ plutôt que $(\alpha\beta)$.

Chapitre 3
Logique pour programmes séquentiels

3.1. Introduction

Le lecteur est, nous l'espérons, déjà convaincu de l'applicabilité de la logique à des domaines variés de la connaissance, dont le point commun est sans doute de donner lieu à des raisonnements relativement formels. Les applications de la logique introduites jusqu'ici étaient du ressort des mathématiques. On peut se demander à ce propos si la logique peut être appliquée à des objets non mathématiques. Pour répondre à cette question, qui fait parfois l'objet d'une polémique, il convient d'abord de savoir quand un objet peut être qualifié de mathématique. Il ne suffit pas qu'il soit *défini* de manière précise, il faut encore qu'il soit *représenté* de manière précise. Les mêmes exigences valent pour les *opérations*, c'est-à-dire pour les techniques de combinaison et de transformation associées à l'objet.

Cette acception est très restrictive mais, en pratique, on contourne facilement la restriction. A partir d'un objet concret que l'on souhaite étudier par des méthodes mathématiques, on imaginera un objet mathématique, "idéalisé", obtenu par une simplification souvent radicale de l'objet concret, permettant néanmoins d'attribuer à celui-ci certaines propriétés de l'objet mathématique associé. Cette technique de modélisation donne d'excellents résultats dans des domaines variés. Par exemple, l'assimilation de la Terre, du Soleil et de la Lune à des points matériels auxquels la loi de Newton s'applique permet de déterminer les trajectoires respectives de ces astres avec une très bonne précision.

On pourra donc appliquer la logique à un certain domaine de la connaissance si les objets de ce domaine peuvent être modélisés mathématiquement. Les

programmes d'ordinateur usuels, écrits en style impératif,[1] n'ont pas a priori le statut d'objets mathématiques, mais plutôt celui d'outils spécialisés, servant à mettre en œuvre d'autres outils, les ordinateurs. Il sera cependant assez facile d'associer à un programme réel un programme idéal qui le modélisera adéquatement. Un élément favorable est le fait qu'un programme (réel) est écrit dans un langage formel; il est donc déjà formellement représenté. La signification d'un programme est a priori informelle; un programme induit un certain comportement dans la machine qui l'exécute. Pour formaliser cette signification, il suffira donc de définir une version idéale de la machine, qui fera abstraction de tous les détails matériels inhérents à une machine réelle. Nous pourrons même nous dispenser de considérer les programmes réels pour nous limiter à leurs modèles; la ressemblance est suffisante pour permettre au lecteur d'appréhender la portée des résultats obtenus. En dehors de son intérêt intrinsèque, l'étude logico-mathématique des programmes illustre bien l'inanité du reproche parfois fait à la logique; par le biais de la modélisation, la logique cesse d'être confinée au domaine mathématique et devient une technique universelle.

Même si la modélisation mathématique des programmes ne présente pas de difficulté majeure, on doit remarquer que les programmes (ou plutôt, leurs modèles) auront par rapport aux objets mathématiques habituels une spécificité notable, à savoir leur taille. En effet, alors que les théorèmes, équations, intégrales, etc., dépassent rarement une page, les programmes de plusieurs dizaines de pages ne sont pas rares.

Heureusement, la difficulté inhérente à l'étude d'un objet formel ne dépend pas principalement de la taille de cet objet. Beaucoup de programmes sont très simples, et les erreurs qu'ils contiennent sont dues à l'inattention du rédacteur. L'usage de la logique peut, dans une certaine mesure, contraindre le programmeur à une rigueur suffisante pour éviter les erreurs, sans pour autant inhiber la créativité et la productivité. La logique est également adaptée à la formalisation et à la résolution des difficultés algorithmiques proprement dites.

La taille des programmes suscite une certaine réflexion à propos de la manière dont l'esprit humain appréhende une réalité complexe. Le mécanisme général est la structuration : un objet complexe est vu comme une collection d'entités plus simples, composées d'une certaine manière. Un programme n'échappe pas à cette règle, et le raisonnement à son propos devra être modulaire. Idéalement, la structure logique d'un programme, c'est-à-dire la structure des raisonnements qui sont faits lors de sa conception ou de son analyse, se reflétera dans la structure syntaxique du programme. Un programme ressemblera alors d'assez près à une formule de logique : la signification du tout s'exprimera fonctionnellement en termes de la signification de ses parties.

[1] La programmation *impérative* est basée sur la notion de *commande* ou d'*affectation*. Les langages Fortran, Cobol, Pascal et C, pour ne citer que ceux-là, sont des langages impératifs.

Cette approche est habituellement adoptée en programmation, même si la logique formelle n'est pas explicitement utilisée. Un programme est vu comme un ensemble de "procédures", ou de "modules", chacun ayant sa fonctionnalité propre. Ces modules interagissent selon un schéma bien spécifié. Il est courant, lors de la production de gros programmes, de spécifier d'abord une architecture de modules, puis de spécifier individuellement chaque module. Il est alors possible de les réaliser indépendamment, et la responsabilité de chacun d'eux peut être confiée à des programmeurs différents. Une méthode de spécification et de preuve de programmes devrait respecter ces principes éprouvés, et modéliser un programme comme une hiérarchie d'objets plutôt que comme un ensemble d'instructions.

Le problème de l'application de la logique à la programmation a suscité beaucoup de recherche. Deux outils se sont avérés particulièrement précieux : la logique de Hoare (Hoare 1969, Apt 1981, Apt 1984) et le calcul de Dijkstra (1976). Ces deux notions constitueront la base théorique de ce chapitre.

3.2. Affectation

3.2.1.. *Introduction*

Pour bien comprendre les principes de base de la programmation, il est utile de réfléchir brièvement sur les principes de fonctionnement des ordinateurs et machines à calculer, qui font à présent partie de notre univers quotidien. La calculette la plus élémentaire comporte essentiellement un mécanisme permettant d'évaluer les *expressions arithmétiques* simples, comme $3 + 6$, $4 + 3 - 2$ et $7 * 5$.[2] Des calculettes plus sophistiquées permettent d'évaluer des expressions comportant crochets et parenthèses, comme $2 * [(3 + 5) * (4 - 2)]$. Nous n'étudierons pas en détail le monde des expressions; notons cependant que ce monde comporte un *lexique* (par exemple, 2 et + sont des symboles du lexique, # et ? n'en sont pas) et une *syntaxe* (par exemple, l'assemblage $2 * (3 + 5)$ est une expression, tandis que l'assemblage $*2)5(3+$ n'en est pas une). Les expressions ont une *sémantique*, assimilable à leur valeur (la valeur de $2 * (3 + 5)$ est 16, celle de $2 * [(3 + 5) * (4 - 2)]$ est 32). Enfin, les aspects *pragmatiques*, concernant en particulier les différents procédés d'évaluation, sont également importants. Nous n'approfondissons pas ces points, malgré leur intérêt, parce que le lecteur en a une connaissance peut-être informelle, mais suffisante.

Nous ne considérons explicitement que les expressions arithmétiques, mais l'ordinateur permet aussi la résolution de problèmes non arithmétiques; les données non numériques sont *codées* sous forme numérique. Par exemple, si on

[2] Les informaticiens utilisent en général le symbole * pour représenter la multiplication.

assimile les valeurs de vérité **F** et **V** à 0 et 1, respectivement, la conjonction se réduit à la multiplication, la négation au complément à 1, et ainsi de suite.

L'expérience montre que l'évaluation d'une expression complexe implique la mémorisation de résultats intermédiaires. Une machine à calculer comporte un ensemble de "cases-mémoires", chacune susceptible de contenir un nombre. Par pressions successives de touches, l'utilisateur peut introduire des nombres dans certaines de ces cases, provoquer l'exécution d'opérations dont les arguments sont ces nombres, forcer l'introduction des résultats de ces opérations dans d'autres cases et lire à l'écran ces résultats. Le programme est ici la séquence de pressions de touches. Une machine à calculer programmable peut mémoriser et donc exécuter à la demande les programmes qui lui sont soumis.

Un ordinateur n'est rien d'autre qu'une machine à calculer programmable, pourvue de divers mécanismes destinés à faciliter sa programmation. Il n'est pas utile ici d'entrer dans les détails; il suffit de souligner que, d'un point de vue relativement abstrait, un ordinateur comporte une collection de cases, ou *variables*, susceptibles de contenir des informations, ou *valeurs* (numériques ou autres). La fonction mathématique qui à chaque variable associe la valeur qu'elle contient est l'*état* de l'ordinateur. Un *programme* engendre une succession de changements d'états; son exécution sera représentée par une séquence d'états. Une *affectation* sera un programme élémentaire, dont l'exécution induit un seul changement d'état.

Remarque. Les cases-mémoires sont souvent appelées "variables" parce qu'au cours d'une exécution de programme, leur contenu, ou valeur, est susceptible de varier. Cette appellation est néanmoins trompeuse pour le logicien ou le mathématicien. En particulier, une variable informatique n'est pas quantifiable; elle ne représente pas un élément arbitraire d'un certain ensemble de référence. Une variable informatique se modélise par une constante logique si l'état est fixé. S'il ne l'est pas, la logique temporelle qui sera abordée dans le volume III modélisera les variables informatiques comme des "constantes flexibles". Il n'est pas utile d'approfondir ces points délicats ici. Le seul point important est que, si un état attribue aux variables informatiques x, y et z une valeur, un procédé élémentaire (non décrit ici) permet d'attribuer une valeur à toute expression arithmétique comportant dans ses opérandes des occurrences de ces variables. Si l'état σ vérifie $\sigma(x) = 1$, $\sigma(y) = 3$ et $\sigma(z) = 0$, et si e est l'expression $x*(y+2)$, alors la valeur de e en l'état σ est $1*(3+2)$, c'est-à-dire 5; on écrira souvent $\sigma(e) = 5$.

3.2.2. *Syntaxe d'une affectation*

On représentera une affectation par une écriture du type

(3.1) $\quad (x_1, \ldots, x_n) := (e_1, \ldots, e_n)\,.$

Les parenthèses sont omises quand cela n'induit pas d'ambiguïté.

Le nombre naturel n est généralement petit. Quand n vaut 0, on a une

affectation vide; si n vaut 1, on a une affectation simple et si n excède 1, on a une affectation multiple.

La partie gauche $(x_1, \ldots, x_n)$ de l'affectation 3.1 est une liste de variables distinctes et la partie droite $(e_1, \ldots, e_n)$ est une liste d'expressions de types appropriés: le type de e_i est le type de x_i. Il est commode de voir le type d'une variable comme l'ensemble des valeurs licites qu'elle peut prendre. Dans ce chapitre comme dans la pratique, le cas le plus courant est le type entier (ensemble $\mathbf{Z}$). La figure 3.1 représente quelques affectations usuelles.

$$
\begin{aligned}
&x := 2\,,\\
&x := x + 1\,,\\
&x, y := y, x\,,\\
&x, y := x - y, x + y\,.
\end{aligned}
$$

Figure 3.1 *Quatre affectations typiques*

En pratique, pour des raisons techniques, les entiers trop grands ne seront pas admis. Si la place réservée en mémoire pour un entier est limitée par exemple à quatre octets, les entiers admissibles iront de -2^{31} à 2^{31}, bornes exclues. Ceci est un exemple de divergence entre le programme réel et son modèle mathématique; certains comportements (le plus souvent indésirables) du programme réel ne seront pas prévus par le modèle.

Le remède général au problème de la divergence entre modèle et objet réel consiste soit à affiner le modèle, soit à utiliser d'autres méthodes (observation, expérimentation et simulation) pour étudier les phénomènes liés à la divergence. L'exemple de la mécanique céleste est instructif à cet égard. Modéliser la Terre par un simple point matériel peut paraître abusif, mais permet néanmoins de rendre compte du mouvement de la Terre autour du Soleil avec une bonne précision. Pour étudier les phénomènes liés à la rotation de la Terre, le modèle doit être affiné et il convient d'ajouter à la notion de point matériel celle d'inertie de rotation. Un raffinement supplémentaire sera nécessaire pour rendre compte de la précession des équinoxes: le scalaire représentant l'inertie sera remplacé par le tenseur d'inertie. Enfin, certains phénomènes de moindre amplitude et très irréguliers semblent échapper à la modélisation; le recours à l'observation est alors le seul moyen de les décrire.

Dans le cas de la programmation, l'attitude scientifique sera un peu différente (il y a une seule Terre, dont l'existence n'est pas le fait de l'humanité, alors que l'ensemble des programmes est potentiellement infini, et que de nouveaux programmes sont continuellement créés). On ne souhaite pas affiner le modèle d'un programme, car le raisonnement à son propos deviendrait trop complexe. Recourir à l'observation et à l'expérimentation est fastidieux, aléatoire et intellectuellement peu satisfaisant.

Une troisième voie est possible. On peut changer (temporairement) le programme réel A (ici, ajouter des tests sur les opérandes, garantissant que

le résultat de l'opération sera admissible) de manière à obtenir un programme réel B en meilleure correspondance avec un modèle mathématique simple. On peut alors déduire les restrictions à imposer aux données, de telle sorte que A et B se comportent de manière analogue. Au prix de ces restrictions, déterminées une fois pour toutes et souvent acceptables, on peut utiliser A (pour lequel n'existe pas de modèle simple et fiable) comme si c'était B (pour lequel un tel modèle existe).

3.2.3. *Sémantique opérationnelle d'une affectation*

L'effet d'une affectation est le suivant. Dans un premier temps, les valeurs des expressions apparaissant à droite du symbole d'affectation sont calculées; dans un second temps, ces valeurs sont attribuées aux variables apparaissant à gauche du symbole d'affectation, ce qui induit un nouvel état. Si A est l'affectation (3.1), on dit qu'une variable x est *affectée* ou *altérée* par A si x apparaît à gauche du symbole d'affectation, c'est-à-dire si x est l'une des variables de l'ensemble $\{x_1, \ldots, x_n\}$; x est *évoquée* par A si x intervient dans une expression de la partie droite de A.[3]

Observons que l'ordre dans lequel variables et expressions sont écrites n'a pas d'influence sur la sémantique de l'affectation. Par exemple, les affectations $(x, y) := (a, b)$ et $(y, x) := (b, a)$ sont équivalentes. (Naturellement, si des permutations distinctes sont appliquées à la liste des variables et à la liste des expressions, l'affectation obtenue ne sera généralement pas équivalente à l'affectation initiale.) Notons aussi que l'affectation double $(x, y) := (a, b)$ n'est pas équivalente à la séquence des deux affectations simples $x := a$ et $y := b$. Par exemple, $(x, y) := (y, x)$ réalise la permutation de x et de y, tandis que la séquence $(x := y;\ y := x)$ est équivalente à l'affectation simple $x := y$.[4]

Un point délicat est l'évaluation d'une expression. D'un point de vue mathématique, une expression est une fonction dont la valeur dépend des valeurs des variables apparaissant dans l'expression. En fait, une expression décrit un algorithme permettant le calcul d'une valeur (celle de l'expression) en fonction d'autres valeurs (celles des variables apparaissant dans l'expression). Si ces dernières valeurs sont inadéquates, le programme d'évaluation peut échouer. Un exemple classique est celui de la division. L'exécution de l'affectation $x := y/z$ est "impossible" quand z vaut 0. Pour l'instant, nous ignorerons ce phénomène d'échec; on verra plus loin comment on peut prévenir de telles situations.

---

[3] ou, de façon légèrement plus précise, si la valeur de l'une au moins des expressions dépend de la valeur de la variable x.

[4] L'affectation $y := x$ est sans effet lorsque l'état de départ vérifie $x = y$. On verra cependant plus loin que l'affectation $x := e$ peut induire un état ne vérifiant pas $x = e$.

3.2.4. *Sémantique relationnelle d'une affectation*

Considérons une *mémoire*, c'est-à-dire un ensemble $\mathcal{M}$ de variables de programme typées. Un état de mémoire σ est une application qui attribue à chaque variable x une valeur σ_x appartenant au type de x. Cette application s'étend immédiatement au domaine des expressions définies sur $\mathcal{M}$; on note σ_e la valeur associée à l'expression e quand les variables apparaissant dans e ont les valeurs spécifiées par l'état σ. L'ensemble des états de la mémoire $\mathcal{M}$ est noté $\Sigma_{\mathcal{M}}$, ou simplement Σ.

La sémantique d'une affectation peut être vue comme une relation binaire (en fait, une fonction), qui est vraie pour un couple (σ, ρ) quand l'exécution de l'affectation dans l'état σ conduit à l'état ρ. Dans un contexte où une mémoire comportant les variables $x_1, \ldots, x_n$ est donnée, on a la définition suivante.

Définition 3.1. Si $X := E$ est l'affectation (3.1), la sémantique relationnelle $R(X := E)$ est le sous-ensemble de $\Sigma \times \Sigma$ défini par la condition

$$(\sigma, \rho) \in R(X := E) \quad \equiv \quad [\forall v\,(v \notin X \Rightarrow \rho_v = \sigma_v) \wedge \forall i\,(\rho_{x_i} = \sigma_{e_i})]\,.$$

Cette définition ne tient pas compte des situations d'échec, pour lesquelles l'une au moins des valeurs σ_{e_i} n'est pas définie. On observe que ρ_{x_i} ne peut différer de σ_{x_i} que si la variable x_i appartient à la liste X, c'est-à-dire si la variable x_i est altérée par l'affectation $X := E$.

La relation $R(X := E)$ est normalement une fonction, qui peut être totale (pour $x := 2$ par exemple) ou partielle (pour $x := x/y$). Il est parfois commode de se donner des affectations *non déterministes*, dont la relation associée n'est pas une fonction. Par exemple, l'affectation $x :=?$ modélise un "générateur aléatoire" et a pour effet d'attribuer à la variable x une valeur arbitraire.

Les affectations non déterministes ne sont guère utiles en tant qu'éléments de programmes achevés, mais elles peuvent constituer un moyen de formaliser des étapes du développement d'un programme. Une affectation non déterministe peut formaliser une commande du type "attribuer à x une valeur satisfaisant la propriété $P(x)$", ou encore "modifier x et y en sorte que $Q(x, y, z)$ devienne vrai", etc. Diverses notations ont été proposées pour représenter les commandes de ce type. Nous n'approfondirons pas cet aspect ici, mais nous signalons simplement que les affectations non déterministes s'introduisent aisément dans les méthodes formelles présentées dans les paragraphes suivants.

3.2.5. *Triplet de Hoare*

Bien souvent, on s'intéresse à un ensemble d'états de la mémoire plutôt qu'à un état isolé. Par exemple, si un programme est destiné à calculer la factorielle d'un nombre initialement contenu dans la variable x de type entier, tout état σ tel que $\sigma_x \in \mathbf{N}$ sera a priori un état initial acceptable. Un tel ensemble d'états est généralement infini et n'est donc pas défini en extension. On utilisera plutôt

un langage du premier ordre pour définir des ensembles d'états, c'est-à-dire des éléments de l'ensemble $\mathcal{P}(\Sigma)$ des parties de Σ. Il y a bijection entre $\mathcal{P}(\Sigma)$ et l'ensemble des applications de Σ dans l'ensemble $\{\mathbf{V}, \mathbf{F}\}$: à tout sous-ensemble $A \subset \Sigma$ correspond la fonction δ_A telle que $\delta_A(\sigma) = \mathbf{V}$ si $\sigma \in A$ et $\delta_A(\sigma) = \mathbf{F}$ si $\sigma \notin A$; réciproquement, à toute application booléenne f correspond l'ensemble $\{x : f(x) = \mathbf{V}\}$. D'autre part, une application de Σ dans $\{\mathbf{V}, \mathbf{F}\}$ est un prédicat sur Σ; on peut représenter des prédicats de ce type par des *assertions*, c'est-à-dire des formules logiques dont les seules variables libres sont les éléments de $\mathcal{M}$. Si l'assertion A' représente le sous-ensemble $A \subset \Sigma$, les trois écritures

$$\sigma \in A\ ,\quad \delta_A(\sigma) = \mathbf{V} \text{ et } \sigma \models A'$$

sont équivalentes.[5] Dans la suite, une même notation sera souvent utilisée pour désigner une assertion et l'ensemble d'états qu'elle représente; on écrira A au lieu de A' et on lira "l'assertion A est vraie dans l'état σ". La priorité de l'opérateur $\models$ sera celle de l'opérateur $\in$.

L'usage des assertions est bien commode mais pose cependant un petit problème. Bien souvent, la mémoire $\mathcal{M}$ est finie mais le type de certaines variables est infini. (Dans ce chapitre, nous ne considérerons que le cas dénombrable; le type d'une variable pourra notamment être l'ensemble $\mathbf{Z}$ des entiers, ou encore l'ensemble A^* des mots issus de l'alphabet fini A.) Il devient alors impossible de représenter tous les ensembles d'états par des assertions. Dans le cas particulier des variables de type entier, les assertions seront des formules arithmético-logiques; l'ensemble des assertions est dénombrable, tandis que l'ensemble $\mathcal{P}(\Sigma)$ ne l'est pas. Pour résoudre ce problème, on observe d'abord que chaque état σ (ou, plus exactement, chaque singleton $\{\sigma\} \subset \Sigma$) est représentable par la conjonction (finie)

$$A_\sigma = \bigwedge_{x \in \mathcal{M}} x = \sigma_x\ .$$

---

[5] Le symbole $\models$ est classiquement utilisé pour représenter la relation de conséquence logique. Si A est une formule et si $\mathcal{E}$ est un ensemble de formules, la notation $\mathcal{E} \models A$ signifie que A est conséquence logique de $\mathcal{E}$, c'est-à-dire que tout modèle de $\mathcal{E}$ est un modèle de A. On utilise aussi cette notation dans un sens quelque peu différent : si $\mathcal{S}$ est une structure (pour un langage logique donné) et si A est une formule (de ce même langage), $\mathcal{S} \models A$ signifie que A est vraie dans $\mathcal{S}$, c'est-à-dire que $\mathcal{S}$ est un modèle de A. Cette notation est parfaitement justifiée dans le cas assez fréquent où on peut assimiler la structure $\mathcal{S}$ à l'ensemble des formules vraies dans cette structure. L'usage qui sera fait ici du symbole $\models$ correspondra à ce cas; en effet, un état σ est déterminé par l'ensemble des identités $\{x = \sigma_x : x \in \mathcal{M}\}$. Remarquons aussi que l'énoncé $\mathcal{E} \models A$ n'est pas une formule de logique, mais plutôt une "métaformule" (une formule à propos de formules). Des notations du type $[(\mathcal{E} \models X) \Rightarrow (\mathcal{E}' \models X')]$ sont donc normalement illicites. Dans le présent contexte toutefois, la métaformule $\sigma \models A'$ est sémantiquement équivalente à la formule $\sigma \in A$, ce qui autorise la notation ci-dessus. Soulignons que cette capacité de coder *simplement* les métaformules en formules est une des caractéristiques les plus précieuses de la théorie des ensembles. D'autres théories, moins riches que la théorie des ensembles, ont également cette propriété. Gödel a démontré que c'était le cas pour les théories incluant (explicitement ou non) l'arithmétique. Le codage arithmétique de métaformules est cependant assez compliqué et peu manipulable.

On représentera alors un sous-ensemble S de Σ par l'assertion $A_S = \bigvee_{\sigma \in S} A_\sigma$. Si S est infini, l'assertion A_S n'est pas une formule de la logique classique (même si sa sémantique est claire). Notons qu'il sera parfois possible de construire une formule (classique) dont la sémantique coïncide avec celle de A_S.

Définition 3.2. Soient P et Q deux assertions. On suppose que l'exécution de l'affectation $X := E$ n'induit pas de situation d'échec, quel que soit l'état initial σ tel que $\sigma \models P$. Le *triplet de Hoare*

(3.2) $\quad \{P\}\ X := E\ \{Q\}$

est une abréviation de la formule

$$\forall \sigma \forall \rho\, [(\sigma \models P \wedge (\sigma, \rho) \in R(X := E)) \Rightarrow \rho \models Q]\,.$$

Autrement dit, si l'affectation $X := E$ est exécutée dans un état où P est vrai, alors l'exécution conduit à un état où Q est vrai. Dans ce contexte, l'assertion P est la *précondition* du triplet; l'assertion Q en est la *postcondition.*

Remarques. Il ressort de la définition qu'un triplet dont la précondition est **F** est toujours vrai; de même, un triplet dont la postcondition est **V** est toujours vrai. Dans le même ordre d'idée, renforcer la précondition et/ou affaiblir la postcondition d'un triplet vrai conduit à un triplet vrai.[6] Notons aussi qu'un triplet de Hoare ou, plus exactement, la formule qu'il représente, comporte des variables liées et des variables de programme, mais pas de variable libre; les notions de vérité et de validité coïncident ici.

3.2.6. *Transformations associées à une affectation*

3.2.6.1. *Préconditions*

L'affectation $X := E$ et une assertion quelconque Q déterminent une partition de l'ensemble des assertions ou, de manière équivalente, de l'ensemble $\mathcal{P}(\Sigma)$, en deux classes, regroupant respectivement les assertions P telles que le triplet de Hoare (3.2) est vrai, et les assertions pour lesquelles ce triplet est faux. La première de ces classes sera notée $pre[X := E;\ Q]$; formellement, on pose

Définition 3.3.

$$pre[X := E;\ Q] \;=\; \{\, P :\ \{P\}\ X := E\ \{Q\}\,\}\,.$$

Si on souhaite que l'exécution de l'affectation $X := E$ conduise dans un état satisfaisant l'assertion Q, il faudra garantir que l'état précédant l'exécution satisfait l'une des assertions de l'ensemble $pre[X := E;\ Q]$. Le théorème suivant précise la structure de cet ensemble.

[6] Ce point est formalisé par la *règle de conséquence* énoncée plus loin.

Théorème 3.1. Un ensemble $\mathcal{A}$ d'assertions est inclus dans l'ensemble $pre[X := E; Q]$ si et seulement si la disjonction $\bigvee \mathcal{A}$ des éléments de $\mathcal{A}$ appartient à $pre[X := E; Q]$.

Preuve. Les formules suivantes sont toutes équivalentes :

$$\mathcal{A} \subset pre[X := E; Q],$$

$$\bigwedge_{P \in \mathcal{A}} (P \in pre[X := E; Q]),$$

$$\forall P\,(P \in \mathcal{A} \Rightarrow [\{P\}\, X := E\, \{Q\}]),$$

$$\forall P\,(P \in \mathcal{A} \Rightarrow \forall \sigma \forall \rho\,[(\sigma \models P \wedge (\sigma, \rho) \in R(X := E)) \Rightarrow \rho \models Q]),$$

$$\forall \sigma \forall \rho \forall P\,[(P \in \mathcal{A} \wedge \sigma \models P \wedge (\sigma, \rho) \in R(X := E)) \Rightarrow \rho \models Q],$$

$$\forall \sigma \forall \rho\,[\exists P\,(P \in \mathcal{A} \wedge \sigma \models P \wedge (\sigma, \rho) \in R(X := E)) \Rightarrow \rho \models Q],$$

$$\forall \sigma \forall \rho\,[((\sigma \models \bigvee \mathcal{A}) \wedge (\sigma, \rho) \in R(X := E)) \Rightarrow \rho \models Q],$$

$$\{\bigvee \mathcal{A}\}\, X := E\, \{Q\},$$

$$\bigvee \mathcal{A} \in pre[X := E; Q].$$

Corollaire 3.2. L'ensemble $pre[X := E; Q]$ est un treillis booléen complet[7] dont les bornes sont $\mathbf{F}$ et $\bigvee pre[X := E; Q]$.

Preuve. On sait déjà que l'ensemble $pre[X := E; Q]$ est fermé pour la disjonction; il est aussi fermé pour la conjonction (non vide) car on a

$$\forall P \forall P_1\,([P \in pre[X := E; Q] \wedge (P_1 \Rightarrow P)] \Rightarrow P_1 \in pre[X := E; Q]).$$

En particulier, si $P, P' \in pre[X := E; Q]$, alors $(P \wedge P') \in pre[X := E; Q]$. Les bornes de l'ensemble des préconditions sont la disjonction $\mathbf{F}$ du sous-ensemble vide et la disjonction de l'ensemble $pre[X := E; Q]$ lui-même. Nous allons voir que cette seconde borne suffit à caractériser l'ensemble; on introduit d'abord une notation spéciale pour cette borne.

Définition 3.4. On pose

$$wlp[X := E; Q] =_{def} \bigvee pre[X := E; Q].$$

Théorème 3.3. L'assertion $wlp[X := E; Q]$ est caractérisée[8] par la relation

$$(3.3) \qquad (\{P\}\, X := E\, \{Q\}) \equiv (P \Rightarrow wlp[X := E; Q]),$$

valable pour toute assertion P.

Preuve. Le théorème 3.1 implique que l'assertion $wlp[X := E; Q]$ appartient à l'ensemble $pre[X := E; Q]$; on a donc

$$\{wlp[X := E; Q]\}\, X := E\, \{Q\}.$$

---

[7] Cette notion classique sera approfondie au chapitre 5.

[8] Si $\mathcal{A}$ est l'algèbre booléenne des assertions, on dit que l'ensemble $pre[X := E; Q]$ est l'*idéal principal* engendré par l'assertion $wlp[X := E; Q]$.

D'autre part, vu le corollaire de ce théorème, on a aussi

$$(P \Rightarrow wlp[X := E;\, Q]) \Rightarrow (\{P\}\, X := E\, \{Q\})\,.$$

Enfin, si $\mathcal{A}$ est un ensemble d'assertions, on a

$$\forall P\, [P \in \mathcal{A} \Rightarrow (P \Rightarrow \bigvee \mathcal{A})]\,,$$

d'où, en fixant $\mathcal{A} = pre[X := E;\, Q]$, l'implication

$$(\{P\}\, X := E\, \{Q\}) \Rightarrow (P \Rightarrow wlp[X := E;\, Q])\,.$$

L'assertion $wlp[X := E;\, Q]$ vérifie donc bien la relation indiquée. Supposons maintenant qu'une assertion B vérifie la relation

$$(\{P\}\, X := E\, \{Q\}) \equiv (P \Rightarrow B)\,,$$

pour toute assertion P. En choisissant $P = wlp[X := E;\, Q]$, on obtient $(wlp[X := E;\, Q] \Rightarrow B)$; d'autre part, en choisissant $P = B$, on obtient $\{B\}\, X := E\, \{Q\}$, d'où $(B \Rightarrow wlp[X := E;\, Q])$, ce qui montre que la seule assertion à vérifier la relation (3.3) est bien $wlp[X := E;\, Q]$.

Remarque. Vu le théorème 3.3, l'assertion $wlp[X := E;\, Q]$ est la *plus faible précondition* rendant vrai le triplet de Hoare (3.2); le symbole wlp évoque l'expression anglaise *weakest liberal precondition.* (On verra plus loin la signification du mot "libéral".)

La proposition suivante donne une expression intrinsèque de la plus faible précondition.

Corollaire 3.4. L'assertion $wlp[X := E;\, Q]$ est aussi caractérisée par l'équivalence

$$\forall \sigma\, (\sigma \models wlp[X := E;\, Q] \equiv \forall \rho\, [(\sigma, \rho) \in R(X := E) \Rightarrow \rho \models Q])\,.$$

Preuve. Soient σ un état quelconque et A_σ l'assertion caractérisant cet état (§ 3.2.5). Les formules du développement suivant sont équivalentes :

$$\begin{array}{l}
\sigma \models wlp[X := E;\, Q]\,,\\
A_\sigma \Rightarrow wlp[X := E;\, Q]\,,\\
\{A_\sigma\}\, X := E\, \{Q\}\,,\\
\forall \sigma_1\, \forall \rho\, [(\sigma_1 \models A_\sigma \wedge (\sigma_1, \rho) \in R(X := E)) \Rightarrow \rho \models Q]\,,\\
\forall \sigma_1\, \forall \rho\, [(\sigma_1 = \sigma \wedge (\sigma_1, \rho) \in R(X := E)) \Rightarrow \rho \models Q]\,,\\
\forall \rho\, [(\sigma, \rho) \in R(X := E) \Rightarrow \rho \models Q]\,.
\end{array}$$

Remarque. Les expressions $\sigma_1 \models A_\sigma$ et $\sigma_1 = \sigma$ sont équivalentes quels que soient les états σ et σ_1.

3.2.6.2. *Postconditions*

On considère à nouveau le triplet de Hoare $\{P\}\, X := E\, \{Q\}$, mais cette fois en supposant que la précondition P est fixée. Ceci détermine une nouvelle partition de l'ensemble des assertions en deux classes. On note $post[P;\, X := E]$ la classe des postconditions rendant vrai le triplet. Plus formellement, on a

$$post[P;\ X := E] = \{Q :\ \{P\}\ X := E\ \{Q\}\}\,.$$

Cette classe donne lieu au théorème suivant.

Théorème 3.5. Un ensemble $\mathcal{A}$ d'assertions est inclus dans l'ensemble $post[P;\ X := E]$ si et seulement si $\bigwedge \mathcal{A}$ est un élément de $post[P;\ X := E]$.

Corollaire 3.6. L'ensemble $post[P;\ X := E]$ est un treillis booléen complet dont les bornes sont $\mathbf{V}$ et $\bigwedge post[P;\ X := E]$.
On omet les preuves, qui sont analogues aux preuves du théorème 3.1 et de son corollaire.

Définition 3.5. On pose

$$slp[P;\ X := E]\ =_{def}\ \bigwedge post[P;\ X := E]\,.$$

Remarque. L'assertion $slp[P;\ X := E]$ est donc la *plus forte postcondition* rendant vrai le triplet de Hoare (3.2). (Le symbole *slp* correspond à l'expression anglaise *strongest liberal postcondition.*)

Comme l'opérateur *wlp*, l'opérateur *slp* donne lieu à deux équivalences remarquables, exprimées par les théorème et corollaire suivants.

Théorème 3.7. L'assertion $slp[P;\ X := E]$ est caractérisée[9] par l'équivalence

$$(\{P\}\ X := E\ \{Q\})\ \equiv\ (slp[P;\ X := E]\ \Rightarrow\ Q)\,.$$

Corollaire 3.8. L'assertion $slp[P;\ X := E]$ est caractérisée par l'équivalence

$$\forall \rho\,(\rho \models slp[P;\ X := E]\ \equiv\ \exists \sigma\,[\sigma \models P\ \wedge\ (\sigma, \rho) \in R(X := E)])\,.$$

3.2.6.3. *Résumé*

Les propriétés utiles des opérateurs *wlp* et *slp* et des triplets de Hoare sont résumées dans les corollaires suivants :

Corollaire 3.9. *Règle de réduction :* les trois formules

$$P\ \Rightarrow\ wlp[X := E;\ Q]\,,$$
$$\{P\}\ X := E\ \{Q\}\,,$$
$$slp[P;\ X := E]\ \Rightarrow\ Q$$

sont équivalentes.

Corollaire 3.10. *Règle de conséquence :*

$$\frac{P'\ \Rightarrow\ P\,,\ \{P\}\ X := E\ \{Q\}\,,\ Q\ \Rightarrow\ Q'}{\{P'\}\ X := E\ \{Q'\}}$$

[9] Si $\mathcal{A}$ est l'algèbre booléenne des assertions, l'ensemble $post[P;\ X := E]$ est le *filtre principal* engendré par l'assertion $slp[P;\ X := E]$.

Corollaire 3.11. *Règle de conjonction et disjonction :* les triplets de Hoare

$$\{P_i\}\ X := E\ \{Q_j\}\ ,\quad i \in I\,, j \in J$$

sont tous vrais si et seulement si le triplet

$$\{\bigvee_{i\in I} P_i\}\ X := E\ \{\bigwedge_{j\in J} Q_j\}$$

est vrai. Ces trois corollaires résument les résultats précédents.

A titre d'exemple, voici d'abord un groupe de triplets de Hoare relatifs à l'affectation $x := x/y$ (les variables x et y sont de type entier; le symbole / représente la division entière) :

$$\{x = 5 \wedge y = 2\}\ x := x/y\ \{x = 2 \wedge y = 2\}\,,$$
$$\{y \neq 0\}\ x := x/y\ \{\mathbf{V}\}\,,$$
$$\{\mathbf{F}\}\ x := x/y\ \{x = 5 \wedge y = -2\}\,.$$

Ces triplets sont tous vrais, au contraire des triplets suivants,

$$\{x = 5 \wedge y = 2\}\ x := x/y\ \{x = 3\}\,,$$
$$\{x = x_0 \wedge y \neq 0\}\ x := x/y\ \{x * y = x_0\}\,,$$
$$\{y \neq 0\}\ x := x/y\ \{x = 5 \wedge y = -2\}\,.$$

qui sont tous faux. Voici enfin des exemples de triplets "problématiques", car correspondant à des situations d'échec certaines ou possibles :

$$\{x = 5 \wedge y = 0\}\ x := x/y\ \{x = 2 \wedge y = 2\}\,,$$
$$\{y = y_0\}\ x := x/y\ \{y = y_0\}\,,$$
$$\{\mathbf{V}\}\ x := x/y\ \{\mathbf{V}\}\,.$$

Il est fréquent d'attribuer la valeur "vrai" aux triplets de ce type, pour des raisons de simplicité formelle. Néanmoins, en général, les situations d'échec sont indésirables et constituent des erreurs de programmation.

3.2.7. *Réduction des assertions à la logique classique*

Les développements qui précèdent ont introduit une signification précise pour la notion d'affectation. Dans ce paragraphe, nous allons montrer que la validité (ou la vérité) d'un triplet de Hoare équivaut à la validité d'une formule de la logique classique.

Un premier pas dans ce sens a déjà été fait. Le corollaire 3.9 montre qu'il est possible de remplacer les triplets de Hoare par des formules de la logique ordinaire, enrichie des "opérateurs" *wlp* et/ou *slp*. Observons cependant que ces opérateurs font intervenir des conjonctions ou disjonctions portant sur un nombre infini de termes. Nous utiliserons sans précautions particulières la "logique infinitaire" ainsi obtenue, parce que sa sémantique se déduit immédiatement de la sémantique de la logique classique. De plus, la théorie de ces opérateurs se développe naturellement dans le cadre de la théorie des ensembles présentée au chapitre 2.

Même s'il n'y a pas d'inconvénient de principe à utiliser une logique infinitaire, il est naturel d'identifier les cas où les assertions sont représentables en logique classique. On a à ce propos le théorème suivant.

Théorème 3.12. Si les assertions P et Q appartiennent à la logique classique, et si A est une affectation, alors les assertions $wlp[A;Q]$ et $slp[P;A]$ appartiennent également à la logique classique.

Preuve. Une affectation A partage la mémoire $\mathcal{M}$ en trois parties. En effet, chaque variable de $\mathcal{M}$ peut être altérée par l'affectation, ou évoquée sans être altérée, ou encore n'être ni évoquée ni altérée. On convient de désigner par x l'ensemble des variables altérées, par y l'ensemble des variables évoquées mais non altérées, et par z le reste de la mémoire. En logique classique, une assertion sera une formule n'ayant pas d'autres variables libres que x, y et z (et leurs composantes). L'affectation A peut donc être récrite

$$x := f(x,y)\,,$$

où $f(x,y)$ est une expression appropriée, dont la valeur ne dépend que des valeurs de x et de y. L'équivalence

$$wlp[x := f(x,y)\,;Q(x,y,z)] \;\equiv\; Q(f(x,y),y,z) \tag{3.4}$$

est une simple conséquence de la définition d'une affectation. La formule correspondante pour l'opérateur slp est

$$slp[P(x,y,z)\,;\; x := f(x,y)] \;\equiv\; \exists\, x_0\,[P(x_0,y,z) \wedge x = f(x_0,y)]\,. \tag{3.5}$$

Pour prouver cette équivalence, on observe que les formules suivantes sont équivalentes :

$$\begin{aligned}
&\rho \;\models\; slp[P(x,y,z)\,;\; x := f(x,y)]\,,\\
&\exists\,\sigma\,[\sigma \models P(x,y,z) \wedge (\sigma,\rho) \in R(x := f(x,y))]\,,\\
&\exists\,\sigma\,[P(\sigma_x,\sigma_y,\sigma_z) \wedge (\rho_x,\rho_y,\rho_z) = (f(\sigma_x,\sigma_y),\sigma_y,\sigma_z)]\,,\\
&\exists\,\sigma_x\,[P(\sigma_x,\rho_y,\rho_z) \wedge \rho_x = f(\sigma_x,\rho_y)]\,,\\
&\rho \;\models\; \exists\, x_0\,[P(x_0,y,z) \wedge x = f(x_0,y)]\,.
\end{aligned}$$

Remarque. Une asymétrie apparaît entre l'opérateur wlp et l'opérateur slp. Elle traduit l'asymétrie inhérente de l'affectation, qui détermine les valeurs finales des variables en fonction des valeurs initiales, mais non l'inverse. En général, on utilisera l'opérateur wlp plutôt que l'opérateur slp. Néanmoins, l'usage de l'opérateur slp se simplifie dans trois cas particuliers fréquents : celui où la précondition spécifie les valeurs initiales des variables altérées, celui de l'affectation constante et celui de l'affectation modélisant une permutation sur le domaine des variables altérées. Dans le premier cas, on a successivement

$$\begin{aligned}
&slp[x = x_0 \wedge P(x_0,y,z)\,;\; x := f(x,y)]\,,\\
&\quad\equiv\; \exists\, x_0'\,[x_0' = x_0 \wedge P(x_0',y,z) \wedge x = f(x_0',y)]\,,\\
&\quad\equiv\; [P(x_0,y,z) \wedge x = f(x_0,y)]\,.
\end{aligned}$$

Dans le deuxième cas, on a successivement

$$
\begin{aligned}
& slp[P(x,y,z)\,;\, x := e_0]\,, \\
& \equiv\ \exists\, x_0\, [P(x_0,y,z) \wedge x = e_0]\,, \\
& \equiv\ [\exists\, x_0\, P(x_0,y,z) \wedge x = e_0]\,.
\end{aligned}
$$

Dans le troisième cas, si pour toute valeur acceptable de la variable y la fonction $f_y : x \to f(x,y)$ est une fonction injective, on a

$$
\begin{aligned}
& slp[P(x,y,z)\,;\, x := f(x,y)]\,, \\
& \equiv\ \exists\, x_0\, [P(x_0,y,z) \wedge x = f(x_0,y)]\,, \\
& \equiv\ \exists\, x_0\, [P(x_0,y,z) \wedge x = f_y(x_0)]\,, \\
& \equiv\ P(f_y^{-1}(x),y,z)\,,
\end{aligned}
$$

puisque $f_y^{-1}(x)$ est l'unique valeur possible de x_0. On a par exemple

$$slp[P(x,y,z)\,;\, x := x+y] \ \equiv\ P(x-y,y,z)\,.$$

3.2.8. *Affectation conditionnelle*

En théorie comme en pratique, il n'est guère possible d'utiliser l'affectation comme seule instruction de base en programmation. Tout d'abord, nous avons vu qu'une affectation peut "échouer". Plus précisément, une affectation A est modélisée par une fonction de domaine Σ. Si la fonction n'est pas totale, il existe un ensemble (ou une assertion) $fail(A)$ contenant les états initiaux conduisant à un échec. Par exemple, si A est l'affectation $x := x/y$, où x et y sont des variables de type entier et où / désigne la division entière, l'assertion d'échec sera $y = 0$.

Une deuxième raison est que, même en l'absence de risque d'échec, on peut souhaiter que la transformation liée à une affectation n'ait lieu que si l'état initial vérifie certaines conditions. Le moyen le plus simple de prévenir, dans certains cas, l'exécution d'une affectation A est de la munir d'une *garde*, ou *condition*, C. On obtient ainsi une *affectation conditionnelle*, ou *affectation gardée*, notée

(3.6) $\quad C \longrightarrow A\,.$

La sémantique opérationnelle de l'affectation conditionnelle (3.6) est la suivante: exécuter A si C est vrai. Quand C est faux, l'affectation n'est pas exécutée. Dans un contexte où plusieurs affectations conditionnelles sont prêtes à être exécutées, il suffit que la garde de l'une d'elles soit vraie pour que l'exécution progresse; si, au contraire, une seule affectation conditionnelle est prête à être exécutée, la fausseté de la garde implique l'arrêt de l'exécution.

L'usage des gardes n'est pas limité à la prévention des échecs, et aucune restriction n'est imposée à la garde. En particulier, les affectations conditionnelles $(\mathbf{V} \longrightarrow A)$ et $(\mathbf{F} \longrightarrow A)$ sont parfaitement licites.

Il est intéressant d'étendre aux affectations conditionnelles la théorie introduite pour les affectations. Pour ce faire, on introduit la définition suivante.

Définition 3.6. La sémantique relationnelle de l'affectation conditionnelle est

$$R(C \longrightarrow A) \;=_{def}\; R(A) \cap (C \times \Sigma)\,.$$

(L'ensemble des états où C est vrai se note également C; il est habituel d'assimiler un prédicat à n arguments à l'ensemble des n-uplets de valeurs pour lesquels le prédicat est vrai.)
L'interprétation est simple. Le couple (σ, ρ) est un élément de la relation associée à $(C \longrightarrow A)$ s'il appartient à la relation associée à A et si, en outre, la condition C est vraie en l'état σ. On a donc

$$(\sigma, \rho) \in R(C \longrightarrow A) \;\equiv\; [(\sigma, \rho) \in R(A) \wedge \sigma \models C]\,.$$

La théorie des triplets de Hoare pour les affectations repose entièrement sur la sémantique relationnelle de ces affectations. Cela signifie que les définitions, de même que les propositions (ainsi que leurs preuves), restent valables pour les affectations conditionnelles. Il suffit de remplacer toute occurrence de $R(A)$, où A est $X := E$, par $R(C \longrightarrow A)$. En particulier, la notion de triplet de Hoare et les opérateurs *pre*, *post*, *wlp* et *slp* se généralisent immédiatement aux affectations conditionnelles.

Il est aussi intéressant de ramener le cas des affectations conditionnelles à celui des affectations inconditionnelles. En ce qui concerne les triplets de Hoare, on a le théorème suivant.

Théorème 3.13. Les triplets

$$\{P\}\; C \longrightarrow A\; \{Q\} \qquad \text{et} \qquad \{P \wedge C\}\; A\; \{Q\}$$

sont équivalents.

Preuve. Les formules du développement suivant sont équivalentes:

$$\{P\}\; C \longrightarrow A\; \{Q\}\,,$$

$$\forall \sigma\, \forall \rho\, [(\sigma \models P \wedge (\sigma, \rho) \in R(C \longrightarrow A)) \;\Rightarrow\; \rho \models Q]\,,$$

$$\forall \sigma\, \forall \rho\, [(\sigma \models P \wedge (\sigma, \rho) \in R(A) \wedge \sigma \models C) \;\Rightarrow\; \rho \models Q]\,,$$

$$\forall \sigma\, \forall \rho\, [(\sigma \models (P \wedge C) \wedge (\sigma, \rho) \in R(A)) \;\Rightarrow\; \rho \models Q]\,,$$

$$\{P \wedge C\}\; A\; \{Q\}\,.$$

On voit que $(C \longrightarrow A)$ se comporte comme A quand C est vrai. Par contre, quand C est faux, l'exécution n'a pas lieu et le triplet se réduit à vrai puisque sa précondition est fausse. On en déduit que la notion de triplet de Hoare formalise la *correction partielle*: l'absence d'exécution est admise; seule l'exécution conduisant à un état falsifiant la postcondition correspond à un triplet faux.

En ce qui concerne les opérateurs *wlp* et *slp*, on a le théorème suivant.

Théorème 3.14. Si P et Q sont des assertions et si $(C \longrightarrow A)$ est une affectation conditionnelle, on a

$$(3.7) \qquad wlp[(C \longrightarrow A);\, Q] \;\equiv\; (C \;\Rightarrow\; wlp[A;\, Q])\,,$$

$$(3.8) \qquad slp[P\,;\,(C \longrightarrow A)] \quad \equiv \quad slp[(P \wedge C);\,A]\,.$$

Preuve. Le développement ci-dessous, dans lequel toutes les formules sont équivalentes, établit la première relation.

$$\begin{aligned}
&\sigma \models wlp[(C \longrightarrow A);\,Q]\,,\\
&\forall\,\rho\,[(\sigma,\rho) \in R(C \longrightarrow A) \;\Rightarrow\; \rho \models Q]\,,\\
&\forall\,\rho\,[(\sigma \models C) \;\Rightarrow\; ((\sigma,\rho) \in R(A) \;\Rightarrow\; \rho \models Q)]\,,\\
&\sigma \models C \;\Rightarrow\; \forall\,\rho\,[(\sigma,\rho) \in R(A) \;\Rightarrow\; \rho \models Q]\,,\\
&\sigma \models C \;\Rightarrow\; \sigma \models wlp[A;\,Q]\,,\\
&\sigma \models (C \;\Rightarrow\; wlp[A;\,Q])\,.
\end{aligned}$$

La seconde relation s'établit au moyen d'un développement semblable.

A titre d'exercice, on peut démontrer l'équivalence des formules

$$\begin{aligned}
&P \;\Rightarrow\; wlp[(C \longrightarrow A);\,Q]\,,\\
&\{P\}\;C \longrightarrow A\;\{Q\}\,,\\
&slp[P;\,(C \longrightarrow A)] \;\Rightarrow\; Q
\end{aligned}$$

en se ramenant au cas des affectations inconditionnelles; voici le développement, constitué de formules équivalentes :

$$\begin{aligned}
&P \;\Rightarrow\; wlp[(C \longrightarrow A);\,Q]\,,\\
&P \;\Rightarrow\; (C \;\Rightarrow\; wlp[A;\,Q])\,,\\
&(P \wedge C) \;\Rightarrow\; wlp[A;\,Q]\,,\\
&\{P \wedge C\}\;A\;\{Q\}\,,\\
&\{P\}\;(C \longrightarrow A)\;\{Q\}\,,\\
&\{P \wedge C\}\;A\;\{Q\}\,,\\
&slp[(P \wedge C);\,A] \;\Rightarrow\; Q\,,\\
&slp[P;\,(C \longrightarrow A)] \;\Rightarrow\; Q\,.
\end{aligned}$$

La lettre l, dans les symboles wlp et slp, rappelle la conception "libérale" de la correction de programme, telle qu'elle est formalisée par la logique de Hoare. Il est clair, cependant, qu'une conception plus stricte de la correction est souvent nécessaire. La *correction totale* correspond à cette exigence, et est formalisée par un opérateur noté wp (*weakest precondition*). Par définition, wp et wlp coïncident pour les affectations (non conditionnelles), pour autant que l'on ignore les situations d'échec. Par contre, les extensions de ces opérateurs aux affectations conditionnelles sont définies comme suit.

$$\begin{aligned}
wlp[(C \longrightarrow A)\,;\,Q] \quad &\equiv \quad (C \;\Rightarrow\; wlp[A;Q])\,,\\
wp[(C \longrightarrow A)\,;\,Q] \quad &\equiv \quad (C \wedge wp[A;Q])\,.
\end{aligned}$$

On voit que wp s'identifie à wlp quand C est vrai, et en diffère quand C est faux. Le risque de blocage de l'exécution est admis dans la version libérale, et exclu dans la version stricte.

3.3. Variables structurées

Les données manipulées par un programme sont parfois "atomiques" mais peuvent aussi être structurées. Les exemples les plus fréquents de données structurées sont probablement le *tableau* et l'*enregistrement*. Le tableau est utilisé pour structurer un ensemble de données de même type, tandis que l'enregistrement regroupe des données de types différents.

3.3.1. *Tableaux*

Un tableau de taille n peut être vu comme une fonction dont le domaine est l'ensemble $\{1, 2, \ldots, n\}$ et dont le codomaine est le type commun de ces *composantes*, c'est-à-dire des n variables que le tableau comporte.

Bien souvent, on souhaitera évoquer ou altérer des composantes isolées d'un tableau. La $i$ème composante du tableau A sera notée $A[i]$, tandis que sa valeur sera notée $A(i)$. (Dans la première expression, A est un tableau, c'est-à-dire un objet informatique; dans la seconde expression, A est la fonction associée à ce tableau et $A(i)$ est la valeur de cette fonction en i.)

Remarque. Le nombre i est l'*index* de la composante $A[i]$. L'expression $A[e]$ désigne une composante du tableau A, dans tout état où la valeur de l'expression e est un index (c'est-à-dire, un nombre compris entre 1 et n).

La théorie des affectations s'applique aux affectations concernant les variables structurées comme à celles concernant les variables simples, mais il convient de bien comprendre les notations. Si A est un tableau linéaire de taille n, une affectation élémentaire concernant la variable A s'écrira

$$A[e] := e' .$$

La valeur de l'expression e doit appartenir au domaine de A, c'est-à-dire à l'ensemble $\{1, \ldots, n\}$, tandis que la valeur de e' appartient au codomaine. L'effet de l'affectation est d'altérer la valeur de A en un seul point du domaine, celui désigné par e; la nouvelle image de ce point est la valeur de e'. Cette écriture, quoique commode, a l'inconvénient de masquer le fait que la variable affectée est A, et non une hypothétique variable isolée $A[i]$, pour une certaine valeur de i. Une écriture plus rigoureuse serait

$$A := \lambda i : \text{if } i = e \text{ then } e' \text{ else } A(i) .$$

(Rappelons que $\lambda x : M(x)$ est la fonction qui à x associe $M(x)$; cette expression se réduit à M si $M(x)$ est l'application de la fonction M à l'argument x, mais reste licite quand $M(x)$ est une expression quelconque.)

Cette écriture est parfois abrégée en

$$A := (A; e : e') .$$

En fait, si f est une fonction, on introduit la notation

$$f' = (f;\, e_1 : e'_1,\, e_2 : e'_2, \ldots, e_n : e'_n) .$$

Cette expression dépend de l'état de mémoire dans laquelle elle est évaluée. Sa valeur est une fonction de même domaine que f. Elle est définie dans l'état σ à condition que les valeurs des e_i dans cet état appartiennent au domaine de f. Les valeurs des e'_i doivent appartenir au codomaine de f. En outre, si e_i et e_j ont même valeur, alors e'_i et e'_j doivent aussi avoir même valeur.

Dans ces conditions, la valeur de f', en un élément x de son domaine, et dans l'état σ, est définie comme suit. Si la valeur en σ de l'un des e_i est x, alors la valeur de $f'(x)$ est la valeur du e'_i correspondant; sinon, $f'(x) = f(x)$.

On a alors la règle suivante :

$$wlp\,[\,A[e] := e';\ P(A)] \ \equiv\ P((A; e : e'))\,.$$

Voici immédiatement un exemple d'application de la règle.

$$\begin{array}{l} wlp[A[x] := 0;\ A(1) = 0] \ \equiv \\ (A; x : 0)\,(1) = 0 \ \equiv \\ (\lambda i : \text{if } i = x \text{ then } 0 \text{ else } A(i))\,(1) = 0 \ \equiv \\ (\text{if } 1 = x \text{ then } 0 \text{ else } A(1)) = 0 \ \equiv \\ (1 = x \wedge 0 = 0) \vee (1 \neq x \wedge A(1) = 0) \ \equiv \\ x = 1 \vee A(1) = 0\,. \end{array}$$

Ce résultat est assez intuitif. Si x vaut 1, alors l'affectation est $A[1] := 0$, et on a évidemment

$$wlp[A[1] := 0;\ A(1) = 0] \ \equiv\ \mathbf{V}\,.$$

Si au contraire x ne vaut pas 1 mais, par exemple, 2, on a

$$wlp[A[2] := 0;\ A(1) = 0] \ \equiv\ A(1) = 0\,.$$

En effet, une affectation altérant uniquement la composante $A[2]$ n'a aucun effet sur la composante $A[1]$.

Remarques. En ce qui concerne les tableaux, il convient d'être prudent, et d'éviter les "raccourcis". On pourrait par exemple penser que l'affectation

$$i := A[i]$$

assure la postcondition

$$i = A(i)\,;$$

ce n'est pas le cas, puisque

$$wlp[i := A[i]\,;\ i = A(i)] \ \equiv\ [A(i) = A(A(i))]\,,$$

ce qui ne se réduit pas à $\mathbf{V}$.[10]

L'affectation $(A[e], A[e']) := (f, f')$ est admissible même au départ d'un état satisfaisant $e = e'$, pour peu que cet état satisfasse aussi $f = f'$. En particulier, l'instruction $(A[i], A[j]) := (A[j], A[i])$ est à la base de nombreux programmes de tri; son exécution au départ d'un état vérifiant $i = j$ est sans effet.

[10] Ce point peut dérouter, mais une affectation du type $x := e$ n'assure pas $x = e$ si l'expression e dépend de x; un exemple très fréquent est $x := x + 1$.

3.3.2. *Enregistrements*

Un *enregistrement* permet de structurer des données de types différents, mais se rapportant à une même entité. Par exemple, on peut regrouper des renseignements concernant une personne dans un enregistrement R, dont les *champs* seraient $R.n$ (nom), $R.p$ (prénom) et $R.a$ (âge). D'un point de vue formel, les enregistrements sont moins complexes que les tableaux, car les *identificateurs de champ* (dans le cas de R, les lettres n, p et a), sont des constantes et non des expressions numériques à évaluer. Un enregistrement sera représenté par une fonction dont le domaine est l'ensemble des identificateurs de champ. Le type de chaque champ doit être spécifié.

Si c est un identificateur de champ pour l'enregistrement R, on a la règle

$$wlp\,[\,R.c := e;\ P(R)] \ \equiv\ P((R; c : e))\,,$$

où c est une constante (non susceptible d'être évaluée).

3.3.3. *Affectations multiples et structures composites*

On est souvent amené à utiliser des structures plus complexes que des tableaux ou des enregistrements à un niveau. On pourrait par exemple ajouter à l'enregistrement R introduit plus haut un champ d'identificateur e pour répertorier les enfants d'une personne. La composante $R.e$ serait un tableau dont les composantes seraient elles-même des enregistrements regroupant le prénom et l'âge de l'enfant. La composante $R.e[i].a$ correspondrait à l'âge du $i$ème enfant.

Il est également fréquent d'affecter simultanément plusieurs composantes appartenant à des structures distinctes ou non. Dans l'affectation multiple

$$(x_1, \ldots, x_n) := (e_1, \ldots, e_n)\,,$$

les x_i peuvent être des variables simples, des variables composites ou des composantes de variables composites. Il faut néanmoins remarquer que la condition de cohérence est plus compliquée que dans le cas des variables simples. En effet, les problèmes ne proviennent plus seulement de l'identité éventuelle entre des variables x_i et x_j, mais aussi de l'inclusion d'une variable dans une autre. Considérons par exemple l'affectation double

$$(A[i][j], A[k]) := (u, v)\,,$$

dans laquelle le tableau A peut être vu comme une matrice (tableau à deux dimensions), et où u et v sont des expressions de types appropriés. L'affectation porte, d'une part, sur la composante individuelle $A[i][j]$ (que l'on note le plus souvent $A[i, j]$) et, d'autre part, sur la ligne $A[k]$. Pour que l'affectation soit licite, il faut que i et k aient des valeurs distinctes, ou encore, si $i = k$, que les valeurs de $v[j]$ et u soient égales. Si on souhaite éliminer cette restriction, on utilisera l'affectation conditionnelle

$$i \neq k \ \vee\ v[j] = u \ \longrightarrow\ (A[i][j], A[k]) := (u, v)\,.$$

3.3.4. *Application : réalisation d'une pile*

Nous avons introduit au paragraphe 2.3.1 le concept de pile. Une pile est une structure dynamique d'enregistrement de données, caractérisée par son mode d'accès : si une donnée a est introduite dans une pile p avant une donnée b, la donnée a ne pourra être extraite de p qu'après la donnée b. (Cela correspond bien à l'idée intuitive d'une pile de documents, par exemple.) Nous avons aussi esquissé une technique de réalisation de pile au moyen d'un tableau. La logique de Hoare va maintenant nous permettre de préciser le lien entre la pile (objet mathématique de type fonctionnel) et sa réalisation (objet informatique susceptible d'être altéré).

Il convient d'abord de souligner la différence fondamentale existant entre les objets mathématiques et les objets informatiques. Considérons d'abord une matrice A (mathématique) à une ligne et n colonnes. Les composantes de A peuvent naturellement être enregistrées dans un tableau (informatique) à n composantes, et il est assez normal d'assimiler ce tableau à la matrice, et de lui donner le même nom A. Cette sorte d'assimilation peut être dangereuse. En effet, si une composante de A, disons la troisième, est modifiée (on lui ajoute 1 par exemple), le mathématicien considèrera qu'il est confronté à une autre matrice, qu'il nommera par exemple A'; il pourra écrire

$$A'_2 = A_2\,,\ A'_3 = A_3 + 1\,.$$

Au contraire, l'informaticien admettra le plus souvent que le tableau A est altéré, mais il distinguera deux états du même tableau, sans introduire un nouveau tableau A'. La notion de triplet de Hoare formalise ce point de vue; dans l'écriture

$$\{A(2) = \alpha \wedge A(3) = \beta\}\ A[3] := A[3] + 1\ \{A(2) = \alpha \wedge A(3) = \beta + 1\}\,,$$

l'ancien état du tableau est décrit (partiellement) par la précondition, tandis que le nouvel état est décrit par la postcondition.

A titre d'exemple d'utilisation des tableaux, nous vérifions — partiellement — le fait qu'un tableau A et un index i, convenablement utilisés, constituent une réalisation concrète de la pile. Nous nous limitons au dernier axiome de la pile, récrit sous forme implicative,

$$\neg\ \mathit{empty?}(p)\ \Rightarrow\ \mathit{push}(\mathit{pop}(p), \mathit{top}(p)) = p\,.$$

Le premier point consiste à transformer la forme $\mathit{push}(\mathit{pop}(p), \mathit{top}(p))$ en une séquence de trois opérations : mémoriser le "sommet" de la pile (top), enlever ce sommet (pop), et "charger" la valeur mémorisée sur la pile (push). On note que l'ordre des opérations est important; en particulier, si le sommet est enlevé avant sa mémorisation, il n'est en principe plus accessible. Les trois opérations sont respectivement modélisées par les affectations

1) $x := A[i]\,,$
2) $i := i - 1\,,$
3) $(i, A[i+1]) := (i + 1, x)\,,$

tandis que la condition de non-vacuité de la pile est modélisée par le test

$$C)\ i > 0\,.$$

Désignons par A_0, i_0 et x_0 les valeurs du tableau, de l'index et de la variable de mémorisation au début des opérations; les mêmes lettres indicées par 1, 2 et 3 représentent les valeurs de ces objets après les première, deuxième et troisième opérations, respectivement. Remarquons que le test C impose la condition $i_0 > 0$. On calcule successivement

$$A_1 = A_0\,,\quad i_1 = i_0\,,\qquad x_1 = A_0(i_0)\,;$$
$$A_2 = A_0\,,\quad i_2 = i_0 - 1\,,\quad x_2 = A_0(i_0)\,.$$

La troisième opération ne sera pas exécutée comme telle, mais décomposée en les affectations simples $i := i + 1$ et $A[i] := x$. L'indice b désigne l'état intermédiaire. On calcule successivement

$$A_b = A_0\,,\quad i_b = i_0\,,\quad x_b = A_0(i_0)\,;$$
$$A_3 = A_0\,,\quad i_3 = i_0\,,\quad x_3 = A_0(i_0)\,.$$

En effet, on a $A_3 = (A_b; i_b : x_b) = (A_0; i_0 : A_0(i_0)) = A_0$. On observe que le tableau A n'a en fait jamais été altéré, contrairement à la pile qu'il modélise, conjointement avec l'index i. (Il n'y a pas de contradiction puisque cet index a lui-même varié.) La nécessité de la condition C apparaît clairement: si cette condition n'est pas vérifiée initialement, la deuxième opération attribue à l'index i une valeur négative, donc illicite.[11] On remarque enfin que la valeur de la variable x a non seulement varié, mais reste définitivement altérée. A nouveau, cela ne compromet pas la validité des propositions correspondant aux axiomes de la pile, la variable x ne participant pas à la modélisation de celle-ci, mais seulement à sa réalisation concrète.

3.4. Systèmes formels séquentiels

Nous allons à présent montrer comment on peut construire des programmes au départ d'affectations conditionnelles. Il existe divers moyens de représenter formellement les programmes impératifs mais, traditionnellement, on distingue les formalismes de "bas niveau", qui privilégient le contrôle du flux d'exécution, et les formalismes de "haut niveau", qui mettent en évidence la structure logique du programme. La notion de système formel séquentiel que nous abordons d'abord appartient à la première catégorie; elle formalise le concept bien connu d'organigramme.

[11] Diminuer un index de pile de valeur nulle est l'équivalent informatique d'ôter un dossier d'une pile qui n'en contient plus! En l'absence d'un système de sécurité spécifique, l'informaticien pourrait ne pas s'aviser de l'erreur; c'est une source — parmi d'autres — de la fiabilité parfois médiocre des logiciels.

3.4.1. *Définition*

Considérons d'abord l'organigramme de la figure 3.2, représentant le programme élémentaire de calcul de la factorielle.

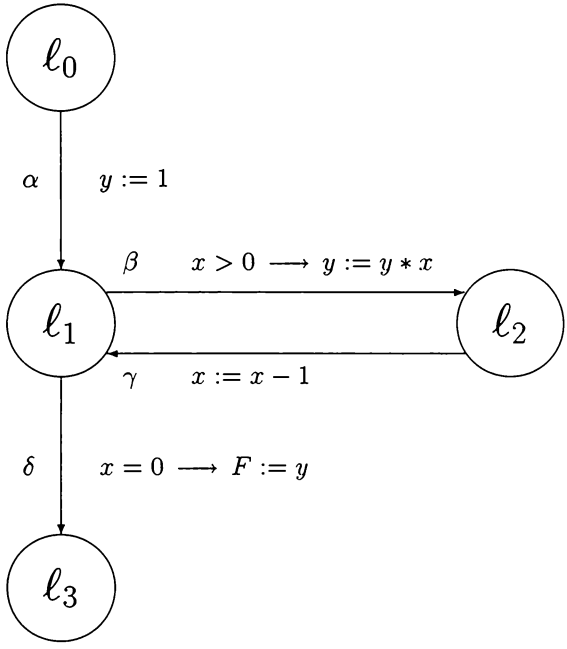

Figure 3.2 *Un exemple d'organigramme*

Une représentation graphique est intuitive, mais devient peu commode si la taille des programmes représentés augmente; de plus, une représentation textuelle se prête mieux à la formalisation.

Un organigramme est un graphe; on peut donc le représenter par une liste de ses nœuds et de ses arcs. On ajoutera aussi une description de la mémoire du programme, c'est-à-dire la liste des variables. L'organigramme donné à la figure 3.2 sera donc décrit par le triplet $(P, \mathcal{M}, \mathcal{T})$, où

$$P = \{\ell_0, \ell_1, \ell_2, \ell_3\};$$

$$\mathcal{M} = \{x, y, F : \textit{entiers}\};$$

$$\begin{aligned}\mathcal{T} = \{\,&\alpha : \ (\ell_0,\ y := 1,\ \ell_1)\,,\\ &\beta : \ (\ell_1,\ x > 0 \longrightarrow y := y * x,\ \ell_2)\,,\\ &\gamma : \ (\ell_2,\ x := x - 1,\ \ell_1)\,,\\ &\delta : \ (\ell_1,\ x = 0 \longrightarrow F := y,\ \ell_3)\,\}\,.\end{aligned}$$

On définit maintenant ce type de représentation de manière formelle. Un *système formel séquentiel* (en abrégé, SFS) comporte trois composants. Le premier est un ensemble fini non vide P dont les éléments sont appelés *étiquettes* ou *points de contrôle*. Le deuxième élément est la *mémoire* $\mathcal{M}$, c'est-à-dire un ensemble fini de variables. A chaque variable est associé un *type*, qui est l'ensemble des valeurs que peut prendre cette variable. Le dernier composant est un ensemble fini et non vide $\mathcal{T}$ de *transitions*. La description et l'étude d'un système formel séquentiel $\mathcal{S} = (P, \mathcal{M}, \mathcal{T})$ implique une série de concepts définis ci-dessous.

Un *état du système* ou, plus brièvement, un *état*, est un couple composé d'une étiquette et d'un état de mémoire. Un *état de mémoire* est une fonction associant une valeur à chaque variable de la mémoire. L'ensemble des états de mémoire est noté Σ et l'ensemble des états du système est noté $\Gamma =_{def} P \times \Sigma$. Un système formel séquentiel peut être vu comme une "machine abstraite"; son fonctionnement donne lieu à une succession de changements d'état.

Une *transition* spécifie une transformation possible de l'état du système; une transition est une liste du type

$$\alpha =_{def} (\ell,\ C \longrightarrow A\ ,\ \ell')\,.$$

L'*origine* ℓ et l'*extrémité* ℓ' sont des étiquettes. La *garde* C est un prédicat interprété sur l'ensemble Σ des états de mémoire et l'*affectation* A dénote une transformation de Σ, définie pour tout état de mémoire satisfaisant la garde. (La notation habituelle sera utilisée pour représenter les affectations.)

Soit s un état du système. La transition α est *exécutable* dans l'état s si s est de la forme (ℓ, σ), avec $\sigma \models C$. L'exécution induit un nouvel état $s' = (\ell', \sigma')$, appelé le α-*successeur* de s. L'état de la mémoire est déterminé par la condition $(\sigma, \sigma') \in R(A)$. Un état s' est un *successeur* de l'état s s'il existe une transition α telle que s' est le α-successeur de s. Un état est *terminal* s'il n'admet pas de successeur.

Une *exécution* est une séquence d'états, finie ou non, telle que chaque état est un successeur de l'état précédent. Le premier état de la séquence est appelé *état initial*. Si l'exécution est finie, le dernier état est appelé *état final*; c'est obligatoirement un état terminal.

Insistons sur le fait qu'une affectation détermine quasi toujours une relation fonctionnelle entre états; en conséquence, chaque transition est déterministe, c'est-à-dire qu'un état admet au plus un α-successeur. Par contre, le système formel séquentiel, considéré globalement, peut être *non déterministe*: un état admet plusieurs successeurs quand plusieurs transitions peuvent être exécutées au départ de cet état.

A titre d'exemple, voici une première exécution possible du système illustré à la figure 3.2.

$$
\begin{aligned}
0 &: (\ell_0, (x = 2, y = -3, F = 17)) \\
1 &: (\ell_1, (x = 2, y = 1, F = 17)) \\
2 &: (\ell_2, (x = 2, y = 2, F = 17)) \\
3 &: (\ell_1, (x = 1, y = 2, F = 17)) \\
4 &: (\ell_2, (x = 1, y = 2, F = 17)) \\
5 &: (\ell_1, (x = 0, y = 2, F = 17)) \\
6 &: (\ell_3, (x = 0, y = 2, F = 2))
\end{aligned}
$$

Cette exécution correspond au calcul de la factorielle de 2; les valeurs initiales de y et F sont sans importance. Voici un second exemple.

$$
\begin{aligned}
0 &: (\ell_0, (x = -2, y = -3, F = 17)) \\
1 &: (\ell_1, (x = -2, y = 1, F = 17))
\end{aligned}
$$

Cette exécution correspond au calcul de la factorielle de -2, non définie.

Remarque. Le formalisme des systèmes formels séquentiels met en évidence les variables et les points de contrôle du programme représenté. Les autres symboles sont supposés avoir une signification indépendante de ce programme. Ces autres symboles sont, d'une part, les signes syntaxiques permettant l'écriture des transitions (parenthèses, flèche, symbole d'affectation, virgule) et, d'autre part, des constantes fonctionnelles et prédicatives appartenant à la signature de théories classiques. Dans le cas du programme de calcul de la factorielle, il s'agit des constantes 0, 1, $-$, $*$ et $>$ de l'arithmétique. Si on ajoute les prédicats de contrôle définis plus loin, on obtient le langage de la théorie du premier ordre associée au programme. Cette théorie s'utilise pour spécifier et vérifier les propriétés des systèmes formels séquentiels (§ 3.4.5).

3.4.2. *Sémantique relationnelle*

On obtient la sémantique relationnelle d'une transition par généralisation de celle d'une affectation. Cette sémantique sera une relation fonctionnelle binaire sur l'ensemble d'états $\Gamma = P \times \Sigma$. Le système formel séquentiel de référence $\mathcal{S} = (P, \mathcal{M}, \mathcal{T})$ est supposé fixé. Soit $\alpha = (\ell,\ C \longrightarrow A,\ \ell')$, une transition de ce système. Un couple $((m, \sigma), (m', \rho))$ d'états du système appartient à la relation $R(\alpha)$ si et seulement si (m', ρ) est le α-successeur de (m, σ). On a donc

Définition 3.7.

$$((m, \sigma), (m', \rho)) \in R(\alpha) \ \equiv\ [m = \ell \wedge m' = \ell' \wedge (\sigma, \rho) \in R(C \longrightarrow A)]\,.$$

On peut aussi associer une sémantique relationnelle au système $\mathcal{S}$ tout entier. Cela revient à donner une sémantique relationnelle au langage de programmation SFS lui-même. Classiquement, la sémantique relationnelle d'un langage de programmation est une loi qui assigne à chaque programme écrit conformément à la syntaxe du langage une relation qui décrit l'effet

de l'exécution de ce programme. Une telle description doit permettre de confirmer ou d'infirmer certaines assertions faites à propos du programme. Dans ce contexte, l'excès d'information est aussi nuisible que le manque d'information. Typiquement, en programmation séquentielle, la relation associée à un programme est la *relation d'entrée-sortie*; elle comporte les couples du type (état initial, état final). Nous nous intéresserons d'abord à une relation plus simple, et nous adoptons la définition suivante.

Définition 3.8.

$$R(\mathcal{S}) =_{def} \bigcup_{\alpha \in \mathcal{T}} R(\alpha) .$$

La sémantique relationnelle d'un système formel séquentiel est donc la réunion des sémantiques relationnelles des transitions de ce système. D'un point de vue opérationnel, un couple d'états (s,t) appartient à la sémantique du système $\mathcal{S}$ s'il existe une transition α du système, exécutable dans l'état s et conduisant à l'état t.

Cette définition, quoique simple, est suffisante parce qu'elle permet de confirmer ou d'infirmer les assertions faites à propos du système $\mathcal{S}$. En particulier, on peut déduire la relation d'entrée-sortie $Z(\mathcal{S})$ de la relation sémantique $R(\mathcal{S})$. On a en effet

$$(s,t) \in Z \;\equiv\; [(s,t) \in R^* \wedge \neg \exists\, u\, ((t,u) \in R)] .$$

Rappelons que la notation R^* désigne la fermeture réflexive et transitive de la relation R; un couple (x,x') appartient à R^* si et seulement s'il existe un entier naturel n et une suite $x_0, \ldots, x_n$ tels que $(x_i, x_{i+1}) \in R$ pour tout $i = 0, \ldots, n-1$ et que $x_0 = x$ et $x_n = x'$. (Le cas $n = 0$ n'est pas exclu : si $n = 0$, alors la suite se réduit à l'unique élément $x = x_0 = x_n = x'$.) D'un point de vue opérationnel, un couple (s,t) appartient à la relation d'entrée-sortie associée au système $\mathcal{S}$ s'il existe une exécution du système dont l'état initial soit s et dont l'état final soit t. Cela implique que l'état t soit terminal.

3.4.3. *Sémantique axiomatique*

Il est naturel d'étendre aux transitions la notion de triplet de Hoare introduite pour les affectations. Pour ce faire, il faut d'abord enrichir le langage des assertions pour permettre la description de la composante "contrôle" d'un état de système.

Définition 3.9. Si le système $\mathcal{S} = (P, \mathcal{M}, \mathcal{T})$ est fixé et si $\ell \in P$, alors le *prédicat de contrôle* associé *at* ℓ (lire "au point de contrôle ℓ", ou simplement "en ℓ") est défini comme suit :

$$(m,\sigma) \models at\ \ell \;\; =_{def} \;\; m = \ell .$$

On introduit aussi la *règle de contrôle*, c'est-à-dire l'axiome

$$\sum_{\ell \in P} at\ \ell \ = \ 1\,.$$

Cette règle implique que, dans tout état du système, un et un seul prédicat de contrôle est vrai.[12]

On définit également l'abréviation suivante. Pour toute assertion A et pour tout point de contrôle ℓ, on note $A[at\ \ell]$ l'assertion obtenue en remplaçant dans A toutes les occurrences de $at\ \ell$ par $\mathbf{V}$ et toutes les occurrences d'autres prédicats de contrôle par $\mathbf{F}$. Le théorème suivant formalise les propriétés de cette assertion.

Théorème 3.15. Quel que soit le point de contrôle ℓ appartenant au processus du système formel $\mathcal{S}$ et quelle que soit l'assertion A relative à ce système, on a

$$\forall \sigma\ ((\ell, \sigma) \models A \ \equiv \ \sigma \models A[at\ \ell])\,,$$
$$at\ \ell \ \Rightarrow \ (A[at\ \ell] \ \equiv \ A)\,,$$
$$\forall \ell_1 \forall \ell_2 \forall \sigma\ ((\ell_1, \sigma) \models A[at\ \ell] \ \equiv \ (\ell_2, \sigma) \models A[at\ \ell])\,.$$

Intuitivement, $A[at\ \ell]$ est obtenu "en rendant $at\ \ell$ vrai dans A". La première propriété paraphrase la définition. La deuxième exprime que $A[at\ \ell]$ est équivalent à A dès que l'on se trouve au point de contrôle ℓ; la troisième propriété exprime que la valeur de vérité de $A[at\ \ell]$ en un état de système ne dépend que de la composante mémoire de cet état.

On peut maintenant étendre aux transitions la notion de triplet de Hoare. Un moyen simple de le faire est d'observer que, dans un programme, la composante de contrôle n'est qu'une variable, ou un ensemble de variables, jouissant d'un statut particulier. Plus concrètement, la transition

$$(\ell,\ C \longrightarrow X := E\,,\ \ell')$$

peut être "simulée" par l'affectation conditionnelle

$$(at\ \ell \wedge C) \longrightarrow (at\ \ell, at\ \ell', X) := (\mathbf{F}, \mathbf{V}, E)$$

si ℓ et ℓ' sont des points de contrôle distincts, et par l'affectation conditionnelle

$$(at\ \ell \wedge C) \longrightarrow X := E$$

si ℓ et ℓ' sont le même point de contrôle.

Cette remarque suggère de définir le triplet

$$\{P\}\ (\ell,\ C \longrightarrow X := E\,,\ \ell')\ \{Q\}$$

comme équivalent au triplet

$$\{P \wedge at\ \ell\}\ C \longrightarrow X := E\ \{Q[at\ \ell'] \wedge at\ \ell\}$$

si ℓ et ℓ' sont distincts, ou au triplet

$$\{P \wedge at\ \ell\}\ C \longrightarrow X := E\ \{Q \wedge at\ \ell\}$$

[12] On utilise la convention habituelle : $\mathbf{F}$ est assimilé à 0 et $\mathbf{V}$ est assimilé à 1.

si ℓ et ℓ' sont identiques. Cette définition n'est guère satisfaisante, parce qu'elle repose uniquement sur une intuition et, surtout, parce que la réduction est imparfaite. En effet, les affectations conditionnelles n'évoquent et n'altèrent que la composante mémoire d'un état de système; on s'attendrait donc à ce que la valeur de vérité de la précondition et de la postcondition des triplets relatifs aux affectations conditionnelles ne dépende aussi que de la composante mémoire de l'état de système dans lequel on se place.

Un moyen d'extension plus rigoureux consiste à partir de la sémantique relationnelle, et à adopter la définition suivante.

Définition 3.10. La formule

$$\{P\}\ \alpha\ \{Q\}$$

est une abréviation de la formule

$$\forall s \forall r\, [(s \models P \wedge (s,r) \in R(\alpha)) \Rightarrow r \models Q]\,.$$

En posant, comme précédemment,

$$\alpha = (\ell,\ C \longrightarrow A\,,\ \ell')\,,$$

et en désignant respectivement par σ et ρ les composantes mémoires des états de système s et r, la formule ci-dessus peut se récrire

$$\forall \sigma \forall \rho\, [[(\ell,\sigma) \models P \wedge ((\ell,\sigma),(\ell',\rho)) \in R(\alpha)] \Rightarrow (\ell',\rho) \models Q)\,,$$

ou encore

$$\forall \sigma \forall \rho\, ((\sigma \models P[at\ \ell] \wedge (\sigma,\rho) \in R(C \longrightarrow A)) \Rightarrow \rho \models Q[at\ \ell'])\,.$$

Ceci suggère la *règle de réduction* suivante : le triplet de Hoare

$$\{P\}\ (\ell,\ C \longrightarrow X := E\,,\ \ell')\ \{Q\}$$

est équivalent au triplet

$$\{P[at\ \ell]\}\ C \longrightarrow X := E\ \{Q[at\ \ell']\}\,.$$

Le théorème suivant montre la convergence des deux approches.

Théorème 3.16. Les triplets de Hoare

$$\{P[at\ \ell]\}\ C \longrightarrow X := E\ \{Q[at\ \ell']\}\,,$$
$$\{P \wedge at\ \ell\}\ C \longrightarrow X := E\ \{Q[at\ \ell']\}$$

et

$$\{P \wedge at\ \ell\}\ C \longrightarrow X := E\ \{Q[at\ \ell'] \wedge at\ \ell\}\,.$$

sont équivalents, quels que soient les points de contrôle ℓ et ℓ', l'affectation conditionnelle $C \longrightarrow X := E$ et les assertions P et Q.

Preuve. Il suffit de vérifier que chaque triplet se réduit à la formule

$$\forall \sigma \forall \rho\, ([\sigma \models P[at\ \ell] \wedge (\sigma,\rho) \in R(C \longrightarrow A)] \Rightarrow \rho \models Q[at\ \ell'])\,.$$

Les triplets de Hoare relatifs aux transitions héritent des propriétés des triplets relatifs aux affectations. C'est, dans chaque cas, une conséquence immédiate des définitions.

La sémantique relationnelle permet également l'extension de la notion de triplet de Hoare au cas où le deuxième argument est un système formel complet. Si $\mathcal{S} = \{P, \mathcal{M}, \mathcal{T}\}$ est un système formel séquentiel, on définira

$$\{P\}\,\mathcal{S}\,\{Q\}$$

comme une abréviation de la formule

$$\forall s \forall r\,[(s \models P \wedge (s,r) \in R(\mathcal{S})) \Rightarrow r \models Q]\,.$$

On a alors le développement suivant, dans lequel toutes les formules sont équivalentes :

$$\begin{aligned}
&\{P\}\,\mathcal{S}\,\{Q\}\,,\\
&\forall s \forall r\,[(s \models P \wedge (s,r) \in R(\mathcal{S})) \Rightarrow r \models Q]\,,\\
&\forall s \forall r\,[(s \models P \wedge (s,r) \in \bigcup_{\alpha\in\mathcal{T}} R(\alpha)) \Rightarrow r \models Q]\,,\\
&\forall s \forall r\,[(s \models P \wedge \bigvee_{\alpha\in\mathcal{T}} (s,r) \in R(\alpha)) \Rightarrow r \models Q]\,,\\
&\forall s \forall r\,[\bigvee_{\alpha\in\mathcal{T}} (s \models P \wedge (s,r) \in R(\alpha)) \Rightarrow r \models Q]\,,\\
&\forall s \forall r \bigwedge_{\alpha\in\mathcal{T}} [(s \models P \wedge (s,r) \in R(\alpha)) \Rightarrow r \models Q]\,,\\
&\bigwedge_{\alpha\in\mathcal{T}} \forall s \forall r\,[(s \models P \wedge (s,r) \in R(\alpha)) \Rightarrow r \models Q]\,,\\
&\bigwedge_{\alpha\in\mathcal{T}} [\{P\}\,\alpha\,\{Q\}]\,.
\end{aligned}$$

En conclusion, le triplet

$$\{P\}\,\mathcal{S}\,\{Q\}\,,$$

relatif à un système formel séquentiel, se réduit à la conjonction

$$\bigwedge_{\alpha\in\mathcal{T}} [\{P\}\,\alpha\,\{Q\}]\,.$$

de triplets relatifs aux transitions du système.

3.4.4. *Transformations associées*

En étendant la notion de triplet de Hoare aux transitions, on a par le fait même généralisé les opérateurs *pre*, *post*, *wlp* et *slp*. Cette généralisation n'est pas essentielle, comme le montre le théorème suivant.

Théorème 3.17. Si P et Q sont des assertions et si $\alpha = (\ell,\ C \longrightarrow A,\ \ell')$ est une transition, on a :

$$(3.9) \qquad wlp[\alpha\,;\ Q] \quad \equiv \quad (at\ \ell \Rightarrow wlp[(C \longrightarrow A);\ Q[at\ \ell']])\,,$$

$$(3.10) \qquad slp[P\,;\ \alpha] \quad \equiv \quad (at\ \ell' \wedge slp[P[at\ \ell];\ (C \longrightarrow A)])\,.$$

Preuve. Le développement ci-dessous, dans lequel toutes les formules sont équivalentes, établit la première relation. On suppose que $s = (m, \sigma)$ est un état quelconque.

$$
\begin{aligned}
& s \models wlp[\alpha;\, Q]\,, \\
& \forall r\, [(s,r) \in R(\alpha) \;\Rightarrow\; r \models Q]\,, \\
& \forall r\, [((m,\sigma),r) \in R(\alpha) \;\Rightarrow\; r \models Q]\,, \\
& \forall \rho\, [(m = \ell \wedge (\sigma,\rho) \in R(C \longrightarrow A)) \;\Rightarrow\; (\ell',\rho) \models Q]\,, \\
& m = \ell \;\Rightarrow\; \forall \rho\, [(\sigma,\rho) \in R(C \longrightarrow A) \;\Rightarrow\; \rho \models Q[at\ \ell']]\,, \\
& s \models at\ \ell \;\Rightarrow\; \forall \rho\, [(\sigma,\rho) \in R(C \longrightarrow A) \;\Rightarrow\; \rho \models Q[at\ \ell']]\,, \\
& s \models at\ \ell \;\Rightarrow\; \sigma \models wlp[(C \longrightarrow A);\, Q[at\ \ell']])\,, \\
& s \models (at\ \ell \;\Rightarrow\; wlp[(C \longrightarrow A);\, Q[at\ \ell']])\,.
\end{aligned}
$$

La seconde relation s'établit au moyen d'un développement semblable.

Les relations entre les opérateurs *wlp* et *slp* et les triplets de Hoare subsistent car elles sont basées entièrement sur la sémantique relationnelle des triplets considérés. En particulier, les trois formules

$$
\begin{aligned}
& P \;\Rightarrow\; wlp[\alpha;\, Q]\,, \\
& \{P\}\ \alpha\ \{Q\}\,, \\
& slp[P;\, \alpha] \;\Rightarrow\; Q
\end{aligned}
\tag{3.11}
$$

sont équivalentes. Voici, à titre d'exercice, des schémas de preuve de ces résultats :

$$
\begin{aligned}
& P \;\Rightarrow\; wlp[\alpha;\, Q]\,, \\
& P \;\Rightarrow\; (at\ \ell \;\Rightarrow\; wlp[(C \longrightarrow A);\, Q[at\ \ell']])\,, \\
& (P \wedge at\ \ell) \;\Rightarrow\; wlp[(C \longrightarrow A);\, Q[at\ \ell']]\,, \\
& (P[at\ \ell] \wedge at\ \ell) \;\Rightarrow\; wlp[(C \longrightarrow A);\, Q[at\ \ell']]\,, \\
& P[at\ \ell] \;\Rightarrow\; wlp[(C \longrightarrow A);\, Q[at\ \ell']]\,, \\
& \{P[at\ \ell]\}\ (C \longrightarrow A)\ \{Q[at\ \ell']\}\,, \\
& \{P\}\ \alpha\ \{Q\}\,.
\end{aligned}
$$

$$
\begin{aligned}
& \{P\}\ \alpha\ \{Q\}\,, \\
& \{P[at\ \ell]\}\ (C \longrightarrow A)\ \{Q[at\ \ell']\}\,, \\
& slp[P[at\ \ell];\, (C \longrightarrow A)] \;\Rightarrow\; Q[at\ \ell']\,, \\
& (at\ \ell' \wedge slp[P[at\ \ell];\, (C \longrightarrow A)]) \;\Rightarrow\; Q[at\ \ell']\,, \\
& (at\ \ell' \wedge slp[P[at\ \ell];\, (C \longrightarrow A)]) \;\Rightarrow\; Q\,, \\
& slp[P;\, \alpha] \;\Rightarrow\; Q\,.
\end{aligned}
$$

Enfin, en ce qui concerne la généralisation des opérateurs *wlp* et *slp* au cas d'un système formel, on a le théorème suivant.

Théorème 3.18. Si $\mathcal{S}$ est un système formel séquentiel dont $\mathcal{T}$ est l'ensemble des transitions, on a

$$(3.12) \quad wlp[\mathcal{S}\,;\, Q] \equiv \bigwedge_{\alpha\in\mathcal{T}} wlp[\alpha\,;\, Q]\,,$$

$$(3.13) \quad slp[P\,;\, \mathcal{S}] \equiv \bigvee_{\alpha\in\mathcal{T}} slp[P\,;\, \alpha]\,.$$

Une fois de plus, ces extensions respectent l'équivalence des formules

$$P \Rightarrow wlp[\mathcal{S};\, Q]\,,$$
$$\{P\}\,\mathcal{S}\,\{Q\}\,,$$
$$slp[P;\, \mathcal{S}] \Rightarrow Q\,.$$

3.4.5. *Spécification et vérification des systèmes séquentiels*

Les propriétés des programmes se déduisent des propriétés des états qui composent leurs exécutions. Une *propriété d'invariance* est vraie de tous les états d'une exécution, tandis qu'une *propriété de progrès* est vraie d'au moins un état de l'exécution (dans le cas d'une exécution finie, ce sera en général le dernier état). Bien souvent, seules les exécutions dont l'état initial satisfait une certaine condition, dite *condition initiale*, sont considérées.

Spécifier un programme consiste à énumérer les propriétés d'invariance et de progrès qu'il doit vérifier. La spécification peut concerner un programme non encore écrit. Quand le programme est écrit, la *vérification* permet de contrôler sa conformité avec les spécifications.

Puisque les programmes sont modélisés par des "générateurs d'exécutions", et que les exécutions sont des trajectoires dans l'espace d'état, la spécification et la vérification des programmes se ramèneront à l'étude des trajectoires dans l'espace d'état. Les techniques de géométrie analytique peuvent s'appliquer à la programmation. La principale différence est qu'en programmation l'espace de référence est discret, alors qu'il est le plus souvent continu en géométrie.

A titre d'exemple, nous calculons les équations paramétriques et cartésienne de la trajectoire associée à une exécution quelconque du système représenté à la figure 3.2. Une trajectoire de ce système est représentée à la figure 3.3; le symbole $\perp$ représente une valeur arbitraire.[13] Les équations paramétriques correspondent au contrôle et à chacune des trois variables du système. Elles sont données à la figure 3.4; le domaine des fonctions $x(.)$, $y(.)$ et $F(.)$ est $\{0, 1, \ldots, 2x_0 + 2\}$; l'élimination du paramètre t donne lieu à l'équation cartésienne de la trajectoire (figure 3.5).

L'équation (3.14) décrit de manière claire et concise le comportement du programme. Par exemple, elle indique que, au point de contrôle ℓ_1, la variable y

[13] On précisera ce point au chapitre 5.

t	C	x	y	F
0	ℓ_0	x_0	$\bot$	$\bot$
1	ℓ_1	x_0	1	$\bot$
2	ℓ_2	x_0	x_0	$\bot$
3	ℓ_1	$x_0 - 1$	x_0	$\bot$
4	ℓ_2	$x_0 - 1$	$x_0 * (x_0 - 1)$	$\bot$
5	ℓ_1	$x_0 - 2$	$x_0 * (x_0 - 1)$	$\bot$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2k+1$	ℓ_1	$x_0 - k$	$x_0 * (x_0 - 1) * \cdots * (x_0 - k + 1)$	$\bot$
$2k+2$	ℓ_2	$x_0 - k$	$x_0 * (x_0 - 1) * \cdots * (x_0 - k)$	$\bot$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2x_0$	ℓ_2	1	$x_0 * (x_0 - 1) * \cdots * 1$	$\bot$
$2x_0 + 1$	ℓ_1	0	$x_0!$	$\bot$
$2x_0 + 2$	ℓ_3	0	$x_0!$	$x_0!$

Figure 3.3 *Trajectoire associée à une exécution*

$$
\begin{array}{l}
(at\ \ell_0)(0)\,, \\
(0 \leq k \leq x_0) \ \Rightarrow\ at\ \ell_1(2k+1)\,, \\
(0 < k \leq x_0) \ \Rightarrow\ at\ \ell_2(2k)\,, \\
(at\ \ell_3)(2x_0+2)\,, \\
x(0) = x_0\,, \\
x(2t+1) = x(2t+2) = x_0 - t\,, \\
y(1) = 1\,, \\
y(2t+2) = y(2t+3) = x_0 * (x_0-1) * \cdots * (x_0 - t) = x_0!/(x_0 - t - 1)!\,, \\
F(2x_0+2) = x_0!\,.
\end{array}
$$

Figure 3.4 *Equations paramétriques d'une trajectoire*

$$
\begin{array}{lll}
 & [\,at\ \ell_0 & \Rightarrow\ x = x_0\,] \\
\wedge & [\,at\ \ell_1 & \Rightarrow\ (0 \leq x \leq x_0 \wedge y * x! = x_0!)\,] \\
\wedge & [\,at\ \ell_2 & \Rightarrow\ (0 < x \leq x_0 \wedge y * (x-1)! = x_0!)\,] \\
\wedge & [\,at\ \ell_3 & \Rightarrow\ (x = 0 \wedge F = y = x_0!)\,]\,.
\end{array}
\tag{3.14}
$$

Figure 3.5 *Invariant, ou équation cartésienne d'une trajectoire*

doit encore être multipliée par $x!$, c'est-à-dire, par x, $x-1$, ..., 2, 1, pour aboutir à la valeur cherchée $x_0!$. Dans le contexte de la programmation, une formule de ce type s'appelle un *invariant*, parce qu'elle établit une relation entre les valeurs des variables du programme qui reste vraie tout au long de l'exécution. Les invariants sont, à l'heure actuelle, le meilleur outil formel de

vérification et de documentation de programme.

L'invariant (3.14) est associé à la condition initiale

$$at\ \ell_0\ \wedge\ x = x_0\ \wedge\ x_0 \in \mathbf{N}\,.$$

Il permet de déduire les propriétés d'invariance du système, et en particulier la correction partielle. Cette propriété exprime que l'exécution ne peut se terminer que sur un état "correct", c'est-à-dire tel que le résultat requis se trouve dans certaines variables. Dans le cas présent, la condition de terminaison est exprimée par l'assertion

(3.15) $\quad at\ \ell_3\ \vee\ (at\ \ell_1\ \wedge\ x < 0)$

qui, vu l'invariant, se réduit à $at\ \ell_3$, tandis que la condition de correction est exprimée par l'assertion $F = x_0!$. La correction partielle est donc modélisée par l'assertion

(3.16) $\quad (at\ \ell_3\ \Rightarrow\ F = x_0!)$

qui doit être vraie à tout instant de l'exécution.

Le lecteur vérifiera sans peine que l'assertion (3.16) est une conséquence logique de l'invariant (3.14); cela suffit à établir la correction partielle du programme.

La situation est moins simple en ce qui concerne la correction totale du programme. L'élimination du paramètre t des équations paramétriques a donné lieu à l'élimination de "l'information temporelle"; il n'est pas possible de déduire de l'invariant le fait que le programme se termine. Tant la terminaison que la correction totale d'un programme sont des propriétés de progrès. Nous considérerons ces propriétés uniquement dans le cadre des programmes structurés (§ 3.5.5).

Nous avons vu que les équations cartésiennes des trajectoires permettent de déterminer les propriétés d'invariance d'un programme. Il est cependant fréquent que l'exacte détermination de cette équation soit très compliquée. Dans un tel cas, on recourra à une notion plus générale d'invariant, donnée par la définition suivante.

Définition 3.11. Un ensemble d'états, ou l'assertion qui le représente, est un *invariant* d'un système si toute trajectoire qui pénètre dans cet ensemble n'en sort pas. Autrement dit, l'assertion I est un invariant du programme $\mathcal{S}$ si la formule suivante est vraie :

(3.17) $\quad \forall\, s,t\,[(s \models I\ \wedge\ (s,t) \in R(\mathcal{S}))\ \Rightarrow\ t \models I]\,.$

La preuve du théorème suivant est immédiate.

Théorème 3.19. L'assertion I est un invariant du système $\mathcal{S}$ si et seulement si le triplet $\{I\}\,\mathcal{S}\,\{I\}$ est vrai.

A titre d'exemple, considérons le système formel de la figure 3.6.

Ce système formel séquentiel calcule le plus grand commun diviseur de trois nombres naturels strictement positifs. Plus précisément, si les valeurs initiales

$$
\begin{aligned}
&P = \{\ell\}\,,\ \ \mathcal{M} = \{x, y, z : \text{entiers}\}\,,\\
&\mathcal{T} = \{\,(\ell,\ x > y \longrightarrow x := x - y,\ \ell)\,,\\
&\qquad\ (\ell,\ y > z \longrightarrow y := y - z,\ \ell)\,,\\
&\qquad\ (\ell,\ z > x \longrightarrow z := z - x,\ \ell)\,\}\,.
\end{aligned}
$$

Figure 3.6 *Plus grand commun diviseur de trois nombres*

des variables x, y, z sont respectivement les nombres naturels $x_0, y_0, z_0 \in \mathbf{N}_0$, alors la valeur finale commune des trois variables sera $pgcd(x_0, y_0, z_0)$.

Le système représenté à la figure 3.6 est très simple, et basé sur un résultat attribué à Euclide : le plus grand commun diviseur de deux nombres naturels ne change pas si le plus petit nombre est soustrait du plus grand. Néanmoins, les équations cartésiennes des trajectoires de ce système sont très compliquées, essentiellement parce qu'il est non déterministe. En effet, dans une situation où $x > y > z$, deux actions sont possibles : on peut soustraire y de x, ou encore soustraire z de y.

L'étude exhaustive des exécutions d'un programme non déterministe est très délicate, même dans les cas simples. Par exemple, au départ de l'état initial caractérisé par $x_0 = 12$, $y_0 = 8$ et $z_0 = 6$, le programme de la figure 3.6 admet cinq exécutions différentes; la plus courte comporte cinq états mais la plus longue en comporte neuf. Il est par contre immédiat de vérifier que la condition

(3.18) $\quad pgcd(x, y, z) = pgcd(x_0, y_0, z_0)$

reste vraie tout au long de chaque exécution, et qu'elle est un invariant du système. C'est une conséquence de la règle d'Euclide; on a par exemple

$$x > y \ \Rightarrow\ pgcd(x - y, y, z) = pgcd(x, y, z)\,.$$

D'autre part, la condition de terminaison est la conjonction de la négation des gardes, c'est-à-dire l'assertion

$$\neg(x > y) \wedge \neg(y > z) \wedge \neg(z > x)\,,$$

qui se réduit à

$$x \leq y \wedge y \leq z \wedge z \leq x\,,$$

puis encore à

$$x = y = z\,.$$

Comme naturellement $pgcd(a, a, a) = a$ pour tout $a \in \mathbf{N}_0$, les valeurs des variables à l'état final satisferont les égalités

$$x = y = z = pgcd(x, y, z) = pgcd(x_0, y_0, z_0)\,.$$

On peut vérifier que si les trois nombres donnés au départ sont strictement positifs, l'exécution se termine nécessairement. Une propriété d'invariance additionnelle est exprimée par l'assertion

$$(0 < x \leq x_0) \wedge (0 < y \leq y_0) \wedge (0 < z \leq z_0)\,.$$

Remarque. Un invariant exprime nécessairement une propriété d'invariance, mais une assertion peut exprimer une propriété d'invariance sans être un invariant. En fait, une assertion exprime une propriété d'invariance si et seulement si elle est la conséquence logique d'un invariant. Nous prouverons de manière constructive au paragraphe 5.4.2 que si l'assertion A exprime une propriété d'invariance du système formel $\mathcal{S}$, pour les conditions initiales exprimées par l'assertion B, alors il existe un invariant I de $\mathcal{S}$ tel que $B \Rightarrow I$ et $I \Rightarrow A$.

3.5. Programmation séquentielle structurée

Une étape décisive dans le domaine de la programmation a été la définition (à la fin des années 50) du langage de programmation ALGOL. Ce langage présente deux aspects intéressants : d'une part, il comporte divers mécanismes facilitant la composition de programmes et, d'autre part, sa syntaxe et sa sémantique font l'objet de définitions formelles. Quoique de nombreux autres langages structurés aient été (et continuent à être) introduits depuis lors, ALGOL reste un bon outil pour l'apprentissage de la programmation. Nous allons définir maintenant une version très simplifiée de ce langage, pour laquelle la logique de Hoare et le calcul de Dijkstra sont particulièrement simples.

3.5.1. *Vocabulaire*

Les symboles utilisés sont les *constantes*, les *variables*[14] et les *combinateurs*. Les combinateurs sont le symbole d'affectation ":=", le symbole de composition séquentielle ";", le symbole de choix "if-then-else", le symbole d'itération "while-do" et les parenthèses d'instructions "[" et "]" (que l'on écrit aussi "begin" et "end"). Les variables et les constantes sont des ensembles de symboles quelconques mais disjoints et distincts des combinateurs. Les constantes sont des objets mathématiques (nombres, fonctions, relations, etc.). Le plus souvent, la correspondance entre le symbole et l'objet mathématique dénoté est implicite. Ce sera le cas notamment pour des symboles tels que 0, 1.5, $+$, $<$, $\mathbf{V}$, etc. Dans le cas de constantes moins habituelles, une définition explicite sera donnée, par exemple dans le langage de la théorie des ensembles.

A chaque variable est associé un type, c'est-à-dire un ensemble non vide. Des types courants sont les ensembles de nombres, l'ensemble des valeurs de vérité, l'ensemble des mots sur un alphabet fixé, etc. A nouveau, des types non classiques peuvent être définis dans le langage de la théorie des ensembles ou dans tout autre formalisme approprié.

[14] Rappelons que ce mot a des acceptions différentes en logique et en programmation. Il est cependant consacré par l'usage et un peu d'attention suffit à éviter toute confusion.

Une variable n'a pas de signification par elle-même, mais uniquement quand elle est associée à un "état de mémoire", c'est-à-dire à une fonction qui, à chaque variable du programme, attribue une valeur appartenant au type de cette variable.

Les expressions sont des assemblages bien formés de constantes et de variables. Un état de mémoire attribue à chaque expression une valeur ne dépendant que des valeurs attribuées aux variables contenues dans l'expression. Les règles de formation et les règles d'évaluation des expressions ne sont habituellement pas données explicitement; elles se déduisent des variables et des constantes contenues dans l'expression.

3.5.2. *Syntaxe des programmes*

Le langage que nous définissons à présent est, comme le langage SFS, basé sur la notion d'affectation, dont nous rappelons brièvement la définition. Une *affectation simple* est un assemblage $x := e$, où x est une variable et e une expression de même type que x. Une *affectation multiple* est un assemblage $(x_1, \ldots, x_n) := (e_1, \ldots, e_n)$, où n est un entier positif; les x_i sont des variables distinctes et les e_i sont des expressions. Pour tout i, l'expression e_i doit avoir le même type que la variable x_i.

Les programmes sont obtenus au moyen des règles suivantes. On suppose que B est une expression booléenne et que S, S_1 et S_2 sont des programmes.

- Une affectation est un programme.
- $[S_1; S_2]$ est un programme.
- [if B then S_1 else S_2] est un programme.
- [while B do S] est un programme.

Un exemple de programme est

$$F =_{def} [y := 1;\ [\text{while } x \neq 0 \text{ do } [y := y * x;\ x := x - 1]]]\,.$$

On l'écrira plus souvent sous la forme

$y := 1;$
while $x \neq 0$ do
    $y := y * x;$
    $x := x - 1.$

(L'indentation permet d'éviter les parenthèses d'instructions.)

Les constantes de ce programme sont $\neq$, 0, $*$, $-$ et 1; elles ont leur signification habituelle. Par exemple, $\neq$ est une relation binaire dont les deux arguments sont des nombres (ici, des nombres entiers). Par conséquent, la variable x est de type entier (de même que la variable y). Les symboles x et y ne peuvent pas être des constantes, puisqu'ils apparaissent à gauche du symbole d'affectation.

Remarques. On note souvent *skip* un programme qui, comme l'affectation vide ou l'affectation $x := x$, n'a pas d'effet. Sa sémantique relationnelle est la relation identique $\{(\sigma, \sigma) : \sigma \in \Sigma\}$. On écrit aussi [if B then S_1] au lieu de [if B then S_1 else *skip*].

3.5.3. *Sémantique relationnelle des programmes structurés*

On ne rappelle pas ici la sémantique *opérationnelle* de ces programmes, c'est-à-dire le comportement d'un ordinateur (simplifié) qui exécuterait ce programme. Signalons seulement qu'une exécution se représente par une suite d'états de mémoire. (La partie "contrôle" n'apparaît pas.) Pour un programme fixé S, l'ensemble des états (de mémoire) possibles est désigné par Σ_S, ou simplement Σ.

3.5.3.1. *Introduction*

Considérons à nouveau le programme F de calcul de la factorielle introduit plus haut. Les variables sont x et y, de type entier. L'ensemble des états est donc $\mathbf{Z}^2$. La relation sémantique que nous utiliserons sera la relation d'entrée-sortie, qui est

$$\{((x_0, y_0), (x_1, y_1)) \ : \ x_0, y_0, x_1, y_1 \in \mathbf{Z} \ \wedge \ x_0 \geq 0 \ \wedge \ x_1 = 0 \ \wedge \ y_1 = x_0!\}\,.$$

En effet, le couple d'état $((x_0, y_0), (x_1, y_1))$ appartient à la relation si et seulement si l'exécution de F, au départ de l'état (x_0, y_0), se termine dans l'état (x_1, y_1).

Remarques. Cette condition n'est pas équivalente à la validité du triplet de Hoare

$$\{x = x_0 \ \wedge \ y = y_0\}\ F\ \{x = x_1 \ \wedge \ y = y_1\}\,,$$

dont la signification sera définie au paragraphe suivant. On verra en particulier que le triplet

$$\{x = -1 \ \wedge \ y = y_0\}\ F\ \{x = x_1 \ \wedge \ y = y_1\}$$

est valide quelles que soient les valeurs de y_0, x_1 et y_1, alors que le couple $((-1, y_0), (x_1, y_1))$ ne peut appartenir à la relation sémantique.[15]

D'autre part, remarquons que la relation associée à un programme écrit dans le langage élémentaire que nous avons utilisé est en fait une fonction, puisque le langage est déterministe : l'état initial détermine univoquement l'exécution.

3.5.3.2. *Définition de la relation sémantique*

Supposons fixée une collection finie V de variables typées. A cette collection correspond l'ensemble Σ des états possibles de la collection; Σ est le produit

[15] La situation serait différente si la condition $x \neq 0$ était remplacée par la condition $x > 0$.

cartésien des types des variables. Si on se limite aux programmes manipulant ces variables, une relation sémantique sera une relation binaire sur l'ensemble Σ. La relation $R(P)$ associée au programme P sera définie par induction sur la structure du programme P. Rappelons la règle pour l'affectation : si $X = (x_1, \ldots, x_n)$ et $E = (e_1, \ldots, e_n)$ sont des listes appropriées, alors

$$(a, b) \in R((x_1, \ldots, x_n) := (e_1, \ldots, e_n))$$

équivaut à

$$a, b \in \Sigma \wedge \forall v\,[(v \notin X \Rightarrow b_v = a_v) \wedge \forall i\,(v = x_i \Rightarrow b_v = a_{e_i})]\,.$$

Etant donné un état a et une expression e, la notation a_e désigne la valeur de e quand on attribue aux variables présentes dans e les valeurs attribuées par l'état a.

La relation associée au programme $[P; Q]$ est simplement la composition des relations associées à P et à Q. On a donc

$$R(P; Q) = R(Q) \circ R(P) =$$
$$\{(a, b) : a, b \in \Sigma \wedge \exists c\,[c \in \Sigma \wedge (a, c) \in R(P) \wedge (c, b) \in R(Q)]\}\,.$$

Si B est une expression booléenne, la relation associée au programme [if B then P else Q] est l'ensemble

$$(R(P) \circ R(B)) \cup (R(Q) \circ R(\neg B))\,,$$

où $R(B)$ est l'ensemble

$$\{(a, a) : a \in \Sigma \wedge a \models B\}\,.$$

Remarque. Nous ne considérons pas les situations d'échec. En particulier, les expressions logiques sont supposées être toujours évaluables, et les ensembles $R(B)$ et $R(\neg B)$ forment une partition de la relation identique sur Σ.

Le cas de l'itération est un peu plus compliqué. On utilise le fait que les deux programmes

$$\text{while } B \text{ do } S$$

et

$$\text{if } B \text{ then } [S; \text{ while } B \text{ do } S] \text{ else } \mathit{skip}$$

doivent avoir la même relation d'entrée-sortie. Si on désigne par P le premier programme, on obtient

$$R(P) = (R(P) \circ R(S) \circ R(B)) \cup R(\neg B)\,,$$

puisque $R(\mathit{skip})$ est la relation identique sur Σ. On peut considérer cette égalité comme une équation dont l'inconnue est la relation $R(P)$. Il s'agit en fait d'une équation de point fixe (chapitre 5). La relation qui nous intéresse est la plus restrictive des solutions de l'équation, c'est-à-dire le plus petit ensemble de couples qui vérifie l'équation. C'est aussi l'intersection de l'ensemble (non vide) des solutions. Pour l'obtenir, on définit la séquence suivante :

$$R_0(P) =_{def} R(\neg B)\,,$$
$$R_{n+1}(P) =_{def} (R_n(P) \circ R(S) \circ R(B)) \cup R(\neg B)\,.$$

La séquence $\mathcal{C} = (R_0(P), R_1(P), \ldots)$ est croissante dans l'ensemble $\mathcal{P}(\Sigma^2)$ ordonné par inclusion;[16] elle admet donc une meilleure borne supérieure qui est $R(P) = \bigcup_n R_n(P)$.

3.5.4. *Logique de Hoare en programmation structurée*

Dans le contexte des programmes structurés, on appelle *triplet de Hoare* un assemblage du type

$$\{A\}\ S\ \{B\}\,,$$

où S est un programme et où A et B sont des formules dont les seules variables libres sont des variables de programme (en fait, du programme S ou d'un programme dont S est un composant). L'interprétation des constantes apparaissant dans A et B est fixée (explicitement ou non) et compatible avec l'interprétation des constantes apparaissant dans S. En conséquence, chaque état $\sigma \in \Sigma_S$ attribue à A et à B des valeurs de vérité, respectivement notées σ_A et σ_B, ou $\sigma \models A$ et $\sigma \models B$.

Remarque. Les formules A et B peuvent contenir des quantifications, mais les symboles utilisés pour dénoter les variables quantifiées doivent toujours être distincts de ceux dénotant des variables de programme.

Le triplet $\{A\}\,S\,\{B\}$ est *vrai*, ou *valide*, si toute exécution finie de S dont l'état initial rend vraie la formule A se termine dans un état qui rend vraie la formule B.

L'intérêt de cette notion est qu'il est possible d'axiomatiser la logique de Hoare. L'unique schéma d'axiome se rapporte à l'instruction d'affectation :

$$\{A(e_1, \ldots, e_n)\}\ (x_1, \ldots, x_n) := (e_1, \ldots, e_n)\ \{A(x_1, \ldots, x_n)\}\,.$$

Dans ce schéma, $A(x_1, \ldots, x_n)$ désigne une formule et $A(e_1, \ldots, e_n)$ est la formule obtenue en y remplaçant uniformément les x_i par les e_i.

Ce schéma d'axiome modélise la sémantique usuelle de l'affectation telle qu'elle a été définie au paragraphe 3.2; on peut aussi considérer que ce schéma *définit* la sémantique (axiomatique) de l'affectation.

Les règles d'inférence sont les suivantes.

$$\frac{\{A\}\ S_1\ \{B\}\,,\ \{B\}\ S_2\ \{C\}}{\{A\}\ S_1; S_2 \{C\}}$$

$$\frac{\{A \wedge B\}\ S_1\ \{C\}\,,\ \{A \wedge \neg B\}\ S_2\ \{C\}}{\{A\}\ \text{if } B \text{ then } S_1 \text{ else } S_2\ \{C\}}$$

$$\frac{\{I \wedge B\}\ S\ \{I\}}{\{I\}\ \text{while } B \text{ do } S\ \{I \wedge \neg B\}}$$

$$\frac{A \Rightarrow B\,,\ \{B\}\ S\ \{C\}\,,\ C \Rightarrow D}{\{A\}\ S\ \{D\}}$$

[16] On démontre facilement par récurrence sur n que $R_n(P) \subset R_{n+1}(P)$.

Ces règles seront désignées dans la suite par les symboles $R1$, $R2$, $R3$ et $R4$, respectivement. Chacune des trois premières règles axiomatise un mécanisme de construction de programme. Une formule I telle que le triplet $\{I \wedge B\}\, S\, \{I\}$ est vrai est un *invariant du programme* [while B do S]. La quatrième règle, que nous avons déjà rencontrée (§ 3.2.6.3), établit le lien entre les mécanismes de programmation et les théories (classiques) relatives aux objets manipulés par le programme. Si par exemple ces objets sont des nombres entiers, les implications $(A \Rightarrow B)$ et $(C \Rightarrow D)$ seront des vérités de l'arithmétique.
Remarque. On parle de "vérités de l'arithmétique", plutôt que de "théorèmes de l'arithmétique", pour que la logique de Hoare n'hérite pas de l'incomplétude inhérente à de nombreuses théories dont, en particulier, la théorie des nombres.

3.5.5. *Terminaison*

Un programme (séquentiel) S est généralement censé conduire d'un état vérifiant A à un état vérifiant B. Il n'est donc pas suffisant de démontrer la validité du triplet de Hoare $\{A\}\, S\, \{B\}$, car celle-ci ne garantit pas qu'un état final soit atteint : le programme S pourrait en effet "boucler" indéfiniment, c'est-à-dire ne jamais se terminer. Comme nous avons exclu les situations d'échec, le "bouclage" ne peut provenir que d'une boucle, dont le corps serait exécuté un nombre infini de fois. Il n'existe pas d'algorithme permettant de décider si l'exécution d'une boucle est susceptible ou non de durer infiniment. Même dans des cas d'apparence élémentaire, la question peut être délicate. Un exemple bien connu est le suivant :

$\{x = x_0 \wedge x_0 \in \mathbf{N}_0\}$
while $x > 1$ do
    if $even(x)$ then $x := x/2$ else $x := 3x + 1$
$\{x = 1\}$.

Chaque fois que x est pair, sa valeur est diminuée; quand il est impair, sa valeur est augmentée. Au total, on observe expérimentalement que, quelle que soit la valeur de départ (entière et positive) x_0 de x, la valeur finale 1 est toujours atteinte. Dans ce cas-ci, la correction partielle est évidente : la valeur finale ne peut pas être différente de 1. Par contre, il n'est pas évident que cette valeur soit toujours atteinte. La séquence conduisant de x_0 à 1 peut être longue et comporter des nombres nettement plus grands que x_0. On a par exemple

7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1.

Dans les cas se présentant en pratique, il existe néanmoins une technique simple permettant de prouver que l'exécution d'une boucle se termine, et de majorer par une constante (ne dépendant que des conditions initiales) le nombre de fois que le corps de la boucle est exécuté.

Chaque exécution du corps de boucle a pour effet de modifier la valeur de certaines variables, en particulier celles intervenant dans la garde de la

boucle. (Si ce n'est pas le cas, il est certain que le corps de boucle sera exécuté indéfiniment.) Si l'exécution de la boucle se termine, il est généralement possible de construire une expression dépendant des variables du programme, et dont la valeur décroît strictement à chaque exécution du corps de boucle. Si l'ensemble ordonné dans lequel l'expression prend ses valeurs est bien fondé, on est assuré de la terminaison de l'exécution. En effet, un ensemble bien fondé ne comporte pas de suite infinie strictement décroissante (§ 2.2.7). L'expression décroissante est parfois appelée le *variant* de la boucle.

3.5.6. *Un exemple élémentaire*

Nous considérons à nouveau le programme F de calcul de la factorielle, introduit au paragraphe 3.5.2.

3.5.6.1. *Correction partielle*

La correction partielle du programme F s'exprime par le triplet

$$\{x = x_0 \ \wedge \ x_0 \geq 0\}\ F\ \{y = x_0!\}\,.$$

La démonstration de la validité de ce triplet est donnée à la figure 3.7. Cette sorte de preuve est généralement représentée de manière plus concise, sous forme d'annotations ajoutées au programme (figure 3.8).

Remarque. Les preuves de ce type ressemblent fort aux preuves de formules de logique classique dans le cadre d'un système axiomatique donné et, plus généralement, aux preuves de résultats mathématiques quelconques. Remarquons néanmoins que la *structure* d'une preuve de Hoare se déduit immédiatement de la structure du programme. En particulier, on détermine par simple inspection du programme où et quand il convient d'utiliser l'axiome et les règles structurelles $R1$, $R2$ et $R3$. En ce qui concerne la règle $R4$, qui constitue l'interface entre les propriétés des données et celles du programme, on peut facilement prouver que, pour n'importe quel triplet valide, il existe une preuve n'utilisant cette règle qu'une fois.

Les difficultés de ce type de preuve sont de deux sortes. Tout d'abord, chaque boucle nécessite la construction d'un invariant, ce qui n'est pas toujours immédiat. On observe que l'invariant formalise, sous une forme très concise, l'effet et la signification de la boucle à laquelle il se rapporte. La difficulté liée à la construction de l'invariant est inhérente à la programmation elle-même. L'autre type de difficulté est liée aux données manipulées. Le programmeur construisant un programme manipulant des nombres entiers doit connaître l'arithmétique. Il n'est notamment pas possible de concevoir le programme très simple que nous venons de présenter sans connaître les propriétés de la factorielle et de la multiplication. En général, le problème n'est pas la connaissance elle-même (un programmeur se risquera rarement à

$$\{x = x_0 \wedge x_0 \geq 0 \wedge 1 = 1\}\ y := 1$$
$$\{x = x_0 \wedge x_0 \geq 0 \wedge y = 1\} \qquad (Axiome)$$
$$\{x = x_0 \wedge x_0 \geq 0\}\ y := 1\ \{x \geq 0 \wedge y * x! = x_0!\} \qquad (R4)$$
$$\{x > 0 \wedge y * x * (x-1)! = x_0!\}\ y := y * x$$
$$\{x > 0 \wedge y * (x-1)! = x_0!\} \qquad (Axiome)$$
$$\{x > 0 \wedge y * x! = x_0!\}\ y := y * x$$
$$\{x > 0 \wedge y * (x-1)! = x_0!\} \qquad (R4)$$
$$\{x - 1 \geq 0 \wedge y * (x-1)! = x_0!\}\ x := x - 1$$
$$\{x \geq 0 \wedge y * x! = x_0!\} \qquad (Axiome)$$
$$\{x > 0 \wedge y * (x-1)! = x_0!\}\ x := x - 1\ \{x \geq 0 \wedge y * x! = x_0!\} \qquad (R4)$$
$$\{x > 0 \wedge y * x! = x_0!\}\ [y := y * x;\ x := x - 1]$$
$$\{x \geq 0 \wedge y * x! = x_0!\} \qquad (R1)$$
$$\{x \geq 0 \wedge y * x! = x_0! \wedge x \neq 0\}\ [y := y * x; x := x - 1]$$
$$\{x \geq 0 \wedge y * x! = x_0!\} \qquad (R4)$$
$$\{x \geq 0 \wedge y * x! = x_0!\}\ [\text{while } x \neq 0 \text{ do } [y := y * x;\ x := x - 1]]$$
$$\{x \geq 0 \wedge y * x! = x_0! \wedge x = 0\} \qquad (R3)$$
$$\{x \geq 0 \wedge y * x! = x_0!\}\ [\text{while } x \neq 0 \text{ do } [y := y * x;\ x := x - 1]]$$
$$\{y = x_0!\} \qquad (R4)$$
$$\{x = x_0 \wedge x_0 \geq 0\}\ F\ \{y = x_0!\} \qquad (R1)$$

Figure 3.7 *Preuve de correction partielle*

$$\{x = x_0 \wedge x_0 \geq 0\}$$
$$y := 1;$$
$$\{x \geq 0 \wedge y * x! = x_0!\}$$
$$\text{while } x \neq 0 \text{ do}$$
$$\{x > 0 \wedge y * x! = x_0!\}$$
$$y := y * x;$$
$$\{x > 0 \wedge y * (x-1)! = x_0!\}$$
$$x := x - 1$$
$$\{x \geq 0 \wedge y * x! = x_0!\}$$
$$\{y = x_0!\}$$

Figure 3.8 *Preuve abrégée de correction partielle*

écrire un programme manipulant des entités qu'il ne connaît pas) mais plutôt l'explicitation et la formalisation de cette connaissance. Tout le monde "sait" que la multiplication est associative et commutative, même si ces deux mots ne font pas partie du vocabulaire de tout un chacun.

Ce phénomène est souvent à la base d'un certain rejet des méthodes formelles de spécification, de conception et de vérification de programme. Le reproche

n'est pas fondé. En effet, un programme est lui-même un texte formel, explicitant (d'une certaine manière) des propriétés des données. Dans la mesure où la correction d'un programme dépend des propriétés des données manipulées, il est normal que le programmeur doive connaître ces propriétés.

3.5.6.2. *Terminaison*

La non-terminaison éventuelle du programme ne pourrait provenir que d'une exécution indéfiniment répétée du corps de la boucle. On prouve que ce phénomène ne se produit pas en exhibant une expression $e(x,y)$, variant dans un ensemble bien fondé $(E,<)$, et dont la valeur diminue chaque fois que le corps de la boucle est exécuté. On doit donc avoir

$$\{e(x,y) = e_0 \wedge x > 0\}\ y := y * x;\ x := x - 1\ \{e(x,y) < e_0\}\,.$$

L'ensemble E sera l'ensemble $\mathbf{N}$ des entiers naturels, qui est bien fondé pour la relation d'ordre numérique habituelle. Le variant $e(x,y)$ sera simplement x, dont la valeur est toujours un entier naturel, qui diminue (d'une unité) à chaque exécution du corps de la boucle.

Remarque. Dans l'exemple de la factorielle, la valeur du variant x est égale au nombre de fois que le corps de la boucle doit encore être exécuté. En général, le variant indiquera seulement une borne supérieure de ce nombre, de manière directe si l'ensemble bien fondé dans lequel il varie est $\mathbf{N}$, de manière parfois fort détournée sinon. Considérons encore le cas du plus grand commun diviseur de deux entiers strictement positifs x_0 et y_0. Il est calculé par le programme

$\{x = x_0 > 0 \wedge y = y_0 > 0\}$
while $x \neq y$ do
    $\{0 < x \leq x_0 \wedge 0 < y \leq y_0 \wedge x \neq y \wedge pgcd(x,y) = pgcd(x_0,y_0)\}$
    if $x > y$ then $x := x - y$ else $y := y - x$
$\{x = y = pgcd(x_0,y_0)\}$.

Deux variants adéquats sont $x+y$ et $max(x,y)$; ils garantissent la terminaison et en outre une complexité linéaire en pire cas (par exemple quand $x_0 = 1 < y_0$). Une étude plus approfondie est cependant nécessaire pour déterminer "le meilleur" variant $e(x,y)$, dont la valeur coïnciderait avec le nombre de tours de boucle restant à effectuer.

Un phénomène un peu plus gênant apparaît avec la variante plus efficace suivante :

$\{x = x_0 \geq 0 \wedge y = y_0 \geq 0\}$
while $x \neq y \wedge x > 0 \wedge y > 0$ do
    $\{0 < x \leq x_0 \wedge 0 < y \leq x_0 \wedge x \neq y \wedge pgcd(x,y) = pgcd(x_0,y_0)\}$
    if $x > y$ then $x := x$ rem y else $y := y$ rem x
$\{max(x,y) = pgcd(x_0,y_0)\}$.

L'expression y rem x dénote le reste de la division de y par x; rappelons aussi que $pgcd(x,x) = pgcd(x,0) = pgcd(0,x) = x$.

Les mêmes variants $x + y$ et $max(x, y)$ s'appliquent ici, mais ne font pas apparaître le fait important que la complexité est ici logarithmique. Plus précisément, il existe une constante c telle que, pour tous x_0 et y_0, le nombre de fois que le corps de boucle est exécuté n'excède pas $c * \log(x_0 + y_0)$.

3.6. Deux exemples classiques

3.6.1. *Tri par insertion*

Nous présentons maintenant une preuve de correction totale pour un algorithme de tri. On se donne une variable A, représentant un tableau linéaire de taille n, dont le codomaine est l'ensemble des entiers. L'algorithme doit trier les éléments de ce tableau en ordre croissant.

3.6.1.1. *Correction partielle*

La correction partielle de l'algorithme *TRI* est spécifiée par le triplet de Hoare suivant :

$$\{\ equal(A, 1, n, A_0, 1, n)\ \}$$
$$TRI$$
$$\{\ perm(A, 1, n, A_0, 1, n)\ \wedge\ ord(A, 1, n)\ \}\,.$$

Intuitivement, la signification de ce triplet est claire. En l'état initial, les n composantes du tableau A forment un tableau fixé A_0. En l'état final, le tableau A est une permutation du tableau initial A_0, et cette permutation est ordonnée. Formellement, la signification de ce triplet dépend d'informations supplémentaires, parfois implicites. Il faut d'abord reconnaître la nature de chaque symbole. Les mots *equal, perm* et *ord* désignent des constantes prédicatives de la théorie des tableaux linéaires; la signification de ces constantes doit être précisée par une définition formelle ou par des axiomes. En pratique, on se contente souvent d'une définition informelle. Le symbole n désigne un entier naturel arbitraire (le symbole 1 désigne naturellement le nombre 1). Les symboles A et A_0 désignent des tableaux linéaires; il faut préciser que le tableau A est représenté par une variable du programme tandis que le tableau A_0 ne l'est pas.

Les prédicats spécifiques de la théorie des tableaux linéaires sont introduits comme suit.

$equal(A, i, j, B, k, \ell) \equiv (i + \ell = j + k \wedge \forall r\,[0 \leq r \leq j - i \Rightarrow A(i + r) = B(k + r)])$.

$ord(A, i, j) \equiv \forall r, s\,[i \leq r \leq s \leq j \Rightarrow A(r) \leq A(s)]$.

$card(A, i, j, z) = |\{r : [i \leq r \leq j \wedge A(r) = z\}|$.

$perm(A, i, j, B, k, \ell) \equiv \forall u\,[u \in \mathbf{Z} \Rightarrow card(A, i, j, u) = card(B, k, \ell, u)]$.

Il faut également spécifier le type des arguments. Par exemple, dans la forme $card(A, i, j, z)$, A est un tableau linéaire, i et j sont des entiers positifs et z est un entier relatif; de plus, la taille de A est supérieure à j.

On voit que, en pratique, une formalisation complète serait laborieuse; en fait, elle n'est vraiment nécessaire que dans le contexte particulier de la vérification *automatique* des programmes.

Avant de donner l'algorithme, nous allons l'expliquer intuitivement, en utilisant la notion de *sous-tableau*. Si $1 \leq i$ et $j \leq n$, $A[i:j]$ est un sous-tableau de $A[1:n]$; son domaine d'index est $\{k \in \mathbf{N} : i \leq k \leq j\}$; le sous-tableau est vide si $i > j$.
L'idée fondamentale est de trier le tableau progressivement, à partir du début. Dans un certain état intermédiaire de l'exécution, la partie gauche $A[1:i-1]$ du tableau est triée, tandis que la partie droite $A[i:n]$ n'a pas été modifiée. Cette situation est formellement décrite par la formule suivante.

$$\text{Inv-Ext} : \quad \begin{array}{l} 1 \leq i \leq n+1 \ \wedge \ \mathit{perm}(A,1,i-1,A_0,1,i-1) \ \wedge \\ \mathit{ord}(A,1,i-1) \ \wedge \ \mathit{equal}(A,i,n,A_0,i,n)\,. \end{array}$$

On voit que le tri se fera en n étapes. La formule donnée ci-dessus est vraie après que la (i-1)ième étape soit terminée et avant que la $i$ème soit commencée. En particulier, cette formule est vraie initialement (avec $i = 1$) et à la fin de l'exécution (avec $i = n+1$).

La $i$ème étape a pour but d'insérer à la bonne place l'élément $A_0(i)$ du tableau, ce qui augmente d'une unité la partie gauche, triée, du tableau A. Cette mise en place se fait par "décalages vers la droite" des éléments supérieurs à l'élément à insérer. Chaque décalage correspond à une sous-étape. Un état intermédiaire typique sera décrit par la formule suivante.

$$\text{Inv-Int} : \quad \begin{array}{l} 1 \leq j \leq i \leq n \ \wedge \ x = A_0(i) \ \wedge \ \mathit{inf}(x,A,j+1,i) \ \wedge \\ \mathit{perm2}(A,1,j-1,A,j+1,i,A_0,1,i-1) \ \wedge \\ \mathit{ord2}(A,1,j-1,A,j+1,i) \ \wedge \\ \mathit{equal}(A,i+1,n,A_0,i+1,n)\,. \end{array}$$

La forme prédicative $\mathit{inf}(x,A,k,\ell)$ est vraie si x est plus petit que les composants $A(k),\ldots,A(\ell)$ du tableau A; $\mathit{perm2}(A,i,j,B,k,\ell,C,m,n)$ est vrai si la concaténation des sous-tableaux $A(i:j)$ et $B(k:\ell)$ est une permutation du sous-tableau $C(m:n)$. De même, la formule $\mathit{ord2}(A,i,j,B,k,\ell)$ est vraie si la concaténation de $A(i:j)$ et $B(k:\ell)$ donne un tableau ordonné.

La valeur j divise la partie gauche $A(1:i)$ du tableau A en deux morceaux : le sous-tableau $A(j+1:i)$, composé d'éléments avec lesquels l'élément à placer a déjà été comparé, et le sous-tableau $A(1:j-1)$ comportant des éléments n'ayant pas été comparés avec l'élément à placer.

La figure 3.9 décrit l'algorithme et résume la preuve de correction partielle. Seules les assertions principales ont été reprises.

3.6.1.2. *Terminaison*

Des variants adéquats sont $n+1-i$ pour la boucle externe et j pour la boucle interne.

```
i := 1;
{ Inv-Ext ∧ i = 1 }
while i ≤ n do
        { Inv-Ext ∧ i ≤ n }
        x := A[i]; j := i;
        { Inv-Int ∧ j = i }
        while j > 1 ∧ A[j − 1] > x do
                A[j] := A[j − 1]; j := j − 1;
                { Inv-Int }
        A[j] := x; i := i + 1
        { Inv-Ext }
{ Inv-Ext ∧ i = n + 1 }
```

Figure 3.9 *Preuve abrégée de correction partielle*

Remarque. L'algorithme de tri qui vient d'être présenté est très simple mais n'est pas très efficace. On peut en effet vérifier que la longueur d'une exécution est proportionnelle au carré de la taille du tableau à trier. Des algorithmes plus astucieux mais moins simples permettent de trier un tableau de taille n en un temps proportionnel à $n \,.\, \log n$. On a aussi prouvé qu'aucun algorithme général basé sur la comparaison des éléments du tableau à trier ne pouvait fonctionner en un temps proportionnel à la taille du tableau.

3.6.2. *Le programme "Find" de Hoare*

Le programme "Find" que nous allons étudier est probablement le premier algorithme un peu délicat à avoir fait l'objet d'une preuve formelle (Hoare 1971). Le problème à résoudre est voisin de celui du tri. Soient un tableau $A_0[1 : N]$ et un indice $f \in \{1, \ldots, N\}$; il faut, par une succession de permutations de paires d'éléments du tableau, aboutir à une version "prétriée" de A. Plus précisément, on doit avoir

$$\forall p, q\,[(1 \leq p \leq f \leq q \leq N) \Rightarrow (A(p) \leq A(f) \leq A(q))]\,.$$

L'élément $A(f)$ est le pivot : toutes les composantes qui le précèdent lui sont inférieures, toutes les composantes qui le suivent lui sont supérieures. Un moyen simple mais inefficace consiste à trier le tableau. En effet, un tableau trié n'est rien d'autre qu'un tableau prétrié pour toute valeur de f. Hoare a proposé l'algorithme plus efficace représenté à la figure 3.10. Cet algorithme fonctionne en un temps proportionnel à la taille N du tableau. Il est souvent utilisé avec la valeur $f = N/2$ ou $f = (N + 1)/2$; dans ce cas, la valeur finale de $A[f]$ est la *médiane* du tableau A_0 (ou plutôt, du multi-ensemble sous-jacent à ce tableau). L'algorithme permet aussi le calcul de n'importe quel percentile, de l'écart interquartile, et d'autres variables statistiques.

```
begin
  {init}
  {I(1,N)}
  (m,n) := (1,N);
  {I(m,n)}
  while m < n do
    {I(m,n) ∧ m < n}
    begin
      {J(A(f),m,n,m,n)}
      (d,i,j) := (A[f],m,n);
      {J(d,i,j,m,n)}
      while i ≤ j do
        begin
          {J(d,i,j,m,n) ∧ i ≤ j}
          while A[i] < d do {J(d,i+1,j,m,n)} i := i+1;
          {J(d,i,j,m,n) ∧ d ≤ A(i)}
          while d < A[j] do {J(d,i,j-1,m,n)} j := j-1;
          {J(d,i,j,m,n) ∧ A(j) ≤ d ≤ A(i)}
          if i ≤ j
            then{J(d,i,j,m,n) ∧ i ≤ j ∧ A(j) ≤ d ≤ A(i)}
              begin
                (A[i],A[j]) := (A[j],A[i]);
                {J(d,i,j,m,n) ∧ i ≤ j ∧ A(i) ≤ d ≤ A(j)}
                {J(d,i+1,j-1,m,n)}
                (i,j) := (i+1,j-1)
              end
          {J(d,i,j,m,n)}
        end;
      {J(d,i,j,m,n) ∧ i > j}
      if i ≤ f {J(d,i,j,m,n) ∧ j<i≤f} then{I(i,n) ∧ m≤i} m := i;
      if f ≤ j {J(d,i,j,m,n) ∧ f≤j<i} then{I(m,j) ∧ j≤n} n := j;
      if j < f < i {J(d,i,j,m,n) ∧ j<f<i} then{I(f,f)} m,n := f,f
      {I(m,n)}
    end{I(m,n) ∧ m ≥ n}
  {I(f,f)}
  {end}
end.
```

Figure 3.10 *Le programme "Find" de Hoare*

Le programme de Hoare est toutefois moins intéressant par ses fonctionnalités propres que par les difficultés de l'approche formelle de la programmation, qu'il met fort bien en évidence. En effet, ce programme, au premier abord, semble très simple. On est donc désagréablement surpris de découvrir que la preuve de ses propriétés n'est pas évidente. Il serait erroné d'en déduire que l'approche formelle est inutilement fastidieuse. Tout simplement, elle ne laisse dans l'ombre aucun point délicat, et ils sont inhabituellement nombreux dans le cas présent.

3.6.2.1. *Correction partielle*

Ce programme est donné avec des annotations à partir desquelles il est facile d'obtenir une preuve de la correction partielle. Les prédicats *init*, I, J et *end* sont définis à la figure 3.11.

$$
\begin{aligned}
&\mathit{init} : \ 1 \leq f \leq N \\
&I(m,n) : \ 1 \leq m \leq f \leq n \leq N \\
&\qquad \wedge \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow \\
&\qquad\qquad (A(p) \leq A(r) \leq A(q))] \\
&J(d,i,j,m,n) : \ m \leq i \wedge j \leq n \wedge 1 \leq m \leq f \leq n \leq N \\
&\qquad \wedge \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow \\
&\qquad\qquad (A(p) \leq A(r) \leq A(q))] \\
&\qquad \wedge \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))] \\
&\mathit{end} : \ 1 \leq f \leq N \\
&\qquad \wedge \forall p,q\,[(1 \leq p \leq f \leq q \leq N) \Rightarrow (A(p) \leq A(f) \leq A(q))]
\end{aligned}
$$

Figure 3.11 *Définition de prédicats auxiliaires*

La correction partielle du programme "Find" repose sur la démonstration des onze propriétés énumérées à la figure 3.12, par le biais du système axiomatique de Hoare. Ces propriétés sont des formules de la théorie du premier ordre associée au programme; ce sont les *conditions de vérification.*

Le corollaire immédiat de ces conditions est le triplet exprimant la correction partielle du programme, c'est-à-dire

$$\{\mathit{init}\}\ \text{Find}\ \{\mathit{end}\}$$

Nous donnons maintenant un schéma de preuve pour chaque condition de vérification.

Preuve de la propriété 1.
$I(1,N)$ se réduit à $(1 \leq f \leq N) \wedge \forall p,q,r\,[\mathbf{F} \Rightarrow \ldots]$, qui se réduit à *init.*

1. $init \Rightarrow I(1, N)$
2. $[I(m,n) \wedge m < n] \Rightarrow J(A(f), m, n, m, n)$
3. $[J(d,i,j,m,n) \wedge i \leq j \wedge A(i) < d] \Rightarrow J(d, i+1, j, m, n)$
4. $[J(d,i,j,m,n) \wedge d \leq A(i) \wedge d < A(j)] \Rightarrow J(d, i, j-1, m, n)$
5. $\{ J(d,i,j,m,n) \wedge i \leq j \wedge A(j) \leq d \leq A(i) \}$
   $(A[i], A[j]) := (A[j], A[i])$
   $\{ J(d,i,j,m,n) \wedge i \leq j \wedge A(i) \leq d \leq A(j) \}$
6. $[J(d,i,j,m,n) \wedge i \leq j \wedge A(i) \leq d \leq A(j)] \Rightarrow J(d, i+1, j-1, m, n)$
7. $[J(d,i,j,m,n) \wedge j < i \leq f] \Rightarrow [I(i,n) \wedge m \leq i]$
8. $[J(d,i,j,m,n) \wedge f \leq j < i] \Rightarrow [I(m,j) \wedge j \leq n]$
9. $[J(d,i,j,m,n) \wedge j < f < i] \Rightarrow I(f,f)$
10. $[I(m,n) \wedge m \geq n] \Rightarrow I(f,f)$
11. $I(f,f) \Rightarrow end$

Figure 3.12 *Conditions de vérification du programme "Find"*

Preuve de la propriété 2.

$J(A(f), m, n, m, n)$ se réduit à

$$\begin{array}{ll} & m \leq m \wedge n \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < m \wedge n < q \leq n) \Rightarrow (A(p) \leq A(f) \leq A(q))]\,. \end{array}$$

La matrice[17] de la formule de la troisième ligne est une implication dont l'antécédent est faux; cette ligne est donc vraie et peut être omise. Les sous-formules $m \leq m$ et $n \leq n$, à la première ligne, peuvent aussi être omises. On obtient ainsi la formule

$$\begin{array}{ll} & 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))]\,, \end{array}$$

qui n'est rien d'autre que $I(m,n)$.

[17] Rappelons que la matrice d'une formule écrite en forme prénexe est la formule débarrassée de ses quantificateurs.

Preuve de la propriété 3.

$J(d,i,j,m,n) \wedge i \leq j \wedge A(i) < d$ se réduit à

$$\begin{array}{ll} & m \leq i \wedge i \leq j \wedge j \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))] \\ \wedge & A(i) < d. \end{array}$$

Cette formule implique

$$\begin{array}{ll} & m \leq i+1 \wedge j \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i+1 \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))], \end{array}$$

qui n'est rien d'autre que $J(d,i+1,j,m,n)$.

Preuve de la propriété 4.

$J(d,i,j,m,n) \wedge d \leq A(i) \wedge d < A(j)$ se réduit à

$$\begin{array}{ll} & m \leq i \wedge j \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))] \\ \wedge & d \leq A(i) \wedge d < A(j). \end{array}$$

Cette formule implique

$$\begin{array}{ll} & m \leq i \wedge j-1 \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i \wedge j-1 < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))], \end{array}$$

qui n'est rien d'autre que $J(d,i,j-1,m,n)$.

Preuve de la propriété 5.

Introduisons d'abord une règle d'inférence dérivée évidente :

$$\frac{\bigwedge_{i=1}^{n} [\,\{P_i\}\,S\,\{Q_i\}\,]}{\{\bigwedge_{i=1}^{n} P_i\}\,S\,\{\bigwedge_{i=1}^{n} Q_i\}}$$

Nous appliquons cette règle avec les instantiations suivantes :

S : $(A[i], A[j]) := (A[j], A[i])$

n : 3

P_1, Q_1 : $m \leq i \leq j \leq n \wedge 1 \leq m \leq f \leq n \leq N \wedge$
$\forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))]$

P_2, Q_2 : $i \leq j \wedge \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))]$

P_3 : $A(j) \leq d \leq A(i)$

Q_3 : $A(i) \leq d \leq A(j)$

La propriété à prouver est

$$\begin{array}{c}\{\, J(d,i,j,m,n) \wedge i \leq j \wedge A(j) \leq d \leq A(i)\,\} \\ (A[i], A[j]) := (A[j], A[i]) \\ \{\, J(d,i,j,m,n) \wedge i \leq j \wedge A(i) \leq d \leq A(j)\,\}\end{array}$$

c'est-à-dire

$$\begin{array}{c}\{P_1 \wedge P_2 \wedge P_3\} \\ S \\ \{Q_1 \wedge Q_2 \wedge Q_3\}\end{array}$$

et chacun des triplets $\{P_i\}S\{Q_i\}$ $(i = 1, 2, 3)$ est évident.

Preuve de la propriété 6.

$J(d,i,j,m,n) \wedge i \leq j \wedge A(i) \leq d \leq A(j)$ se réduit à

$$\begin{array}{ll} & m \leq i \wedge i \leq j \wedge j \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))] \\ \wedge & A(i) \leq d \wedge d \leq A(j)\,.\end{array}$$

Cette formule implique

$$\begin{array}{ll} & m \leq i+1 \wedge j-1 \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i+1 \wedge j-1 < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))]\,,\end{array}$$

qui est simplement $J(d, i+1, j-1, m, n)$.

Preuve de la propriété 7. (La preuve de la propriété 8 est analogue.)

$J(d,i,j,m,n) \wedge j < i \leq f$ se réduit à

$$\begin{array}{ll} & m \leq i \wedge j < i \leq f \wedge j \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))]\,.\end{array}$$

Cette formule implique

$$\begin{array}{ll} & m \leq i \wedge j < i \wedge 1 \leq i \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq A(q))]\,,\end{array}$$

qui implique

$$\begin{array}{ll} & m \leq i \wedge j < i \wedge 1 \leq i \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,q\,[(m \leq p < i \wedge i \leq q \leq n) \Rightarrow (A(p) \leq A(q))]\,.\end{array}$$

Dans la troisième ligne, la variable liée q est renommée r :

$$\begin{array}{ll} & m \leq i \wedge j < i \wedge 1 \leq i \leq f \leq n \leq N \\ \wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p,r\,[(m \leq p < i \wedge i \leq r \leq n) \Rightarrow (A(p) \leq A(r))]\,.\end{array}$$

Les deuxième et troisième lignes sont recombinées en

$$
\begin{array}{ll}
 & m \leq i \wedge j < i \wedge 1 \leq i \leq f \leq n \leq N \\
\wedge & \forall p,q,r\,[(1 \leq p < i \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))]\,,
\end{array}
$$

qui n'est rien d'autre que $I(i,n) \wedge m \leq i \wedge j < i$.

Preuve de la propriété 9.

$J(d,i,j,m,n) \wedge j < f < i$ se réduit à

$$
\begin{array}{ll}
 & j < f < i \wedge 1 \leq m \leq f \leq n \leq N \\
\wedge & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\
\wedge & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))]\,.
\end{array}
$$

Cette formule implique

$$
\begin{array}{ll}
 & j < f < i \wedge 1 \leq m \leq f \leq n \leq N \\
\wedge & \forall p,q\,[(1 \leq p < m \leq f \leq n < q \leq N) \Rightarrow (A(p) \leq A(f) \leq A(q))] \\
\wedge & \forall p\,[(m \leq p < i \wedge j < f \leq n) \Rightarrow (A(p) \leq A(f))] \\
\wedge & \forall q\,[(m \leq f < i \wedge j < q \leq n) \Rightarrow (A(f) \leq A(q))]\,.
\end{array}
$$

Les deux dernières lignes sont combinées, en tenant compte des deux premières :

$$
\begin{array}{ll}
 & j < f < i \wedge 1 \leq m \leq f \leq n \leq N \\
\wedge & \forall p,q\,[(1 \leq p < m \leq f \leq n < q \leq N) \Rightarrow (A(p) \leq A(f) \leq A(q))] \\
\wedge & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq A(f) \leq A(q))]\,.
\end{array}
$$

Cette formule implique

$$
\begin{array}{ll}
 & j < f < i \wedge 1 \leq m \leq f \leq n \leq N \\
\wedge & \forall p,q\,[(1 \leq p \leq f \leq q \leq N) \Rightarrow (A(p) \leq A(f) \leq A(q))]\,,
\end{array}
$$

qui implique visiblement $I(f,f)$.

Preuve de la propriété 10.

$I(m,n) \wedge m \geq n$ se réduit à

$$
\begin{array}{ll}
 & 1 \leq m = f = n \leq N \\
\wedge & \forall p,q,r\,[(1 \leq p < m = r = f = n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))]\,.
\end{array}
$$

Cette formule se réduit à

$$
\begin{array}{ll}
 & 1 \leq f \leq N \wedge m = f = n \\
\wedge & \forall p,q,r\,[(1 \leq p < r = f < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))]
\end{array}
$$

c'est-à-dire $I(f,f)$.

Preuve de la propriété 11.

$I(f,f)$ est

$$
\begin{array}{ll}
 & 1 \leq f \leq N \wedge m = f = n \\
\wedge & \forall p,q,r\,[(1 \leq p < r = f < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))]
\end{array}
$$

et implique visiblement

$$
\begin{array}{ll}
 & 1 \leq f \leq N \\
\wedge & \forall p,q\,[(1 \leq p < f < q \leq N) \Rightarrow (A(p) \leq A(f) \leq A(q))]
\end{array}
$$

qui est équivalent à

$$1 \leq f \leq N$$
$$\wedge \quad \forall p, q\,[(1 \leq p \leq f \leq q \leq N) \Rightarrow (A(p) \leq A(f) \leq A(q))]$$

qui est la formule *end*.

Cette preuve de correction partielle est assez longue; cela explique le manque de popularité de l'approche formelle en pratique. On observera néanmoins que le travail de démonstration est systématique, et repose entièrement sur la découverte des prédicats I et J. Cette découverte n'est pas évidente, mais celle de l'algorithme lui-même ne l'est pas davantage; les prédicats I et J formalisent l'idée intuitive du programmeur. Apprendre à utiliser l'approche formelle revient à apprendre à *formaliser* son intuition ... et non à s'en passer. La programmation reste actuellement une activité créative.

Leur longueur relative n'est pas le seul point faible des preuves formelles. On doit bien constater que beaucoup de preuves formelles comportent des lacunes. C'est le cas de celle que nous venons de donner: nulle part n'est énoncé ou démontré le fait que $A[x]$ n'est évoqué ou altéré que pour des valeurs licites de x, comprises entre 1 et N (propriété d'indice). Bien sûr, il est possible de pallier cette lacune, au prix d'invariants plus complets. On peut aussi laisser les choses en l'état, en notant qu'un système moderne auquel ce programme serait soumis détecterait, à l'exécution, toute violation éventuelle de cette propriété. Cette attitude est peu satisfaisante pour l'esprit et, en cas de violation détectée, le programmeur devra quand même s'interroger sur la cause de l'erreur. Dans la suite, nous supposerons que la propriété d'indice est vérifiée.

3.6.2.2. *Terminaison*

L'algorithme comporte quatre boucles. La boucle externe a pour garde $m < n$; le corps de cette boucle comporte une boucle moyenne, dont la garde est $i \leq j$. Le corps de celle-ci comporte deux boucles internes, de gardes respectives $A[i] < d$ et $d < A[j]$. La terminaison d'une boucle étant dépendante de la terminaison des boucles plus internes qu'elle contient, il est naturel de considérer d'abord les boucles internes. Elles ne posent d'ailleurs aucun problème si on admet (ou démontre) la propriété d'indice.

On récrit d'abord la première boucle interne, avec les assertions qui la concernent;

$\{J(d,i,j,m,n) \wedge i \leq j\}$
while $A[i] < d$ do $\{J(d,i+1,j,m,n)\}$ $i := i+1$;
$\{J(d,i,j,m,n) \wedge d \leq A(i)\}$

Le variant doit diminuer quand i augmente; i ne peut dépasser N, ce qui suggère le variant $N - i$. La deuxième boucle interne se traite de même, avec le variant j.

La boucle moyenne a pour garde $i \leq j$, ce qui suggère le variant $j - i$. Rien ne garantissant que cette quantité reste un entier naturel, on peut préférer le

variant $N + j - i$, qui ne pourrait être négatif sans violation de la propriété d'indice. Il reste à s'assurer que ce variant décroît d'au moins une unité chaque fois que le corps de la boucle moyenne est exécuté. On observe d'abord que le variant n'augmente jamais. Ensuite, si l'une (au moins) des deux boucles internes est exécutée au moins une fois, le variant décroît effectivement. Sinon, la condition de l'instruction conditionnelle "if" suivant ces deux boucles est vérifiée, ce qui implique aussi la décroissance du variant.

Enfin, la boucle externe a pour garde $m < n$, ce qui suggère le variant $n - m$ (il est facile de voir que cette expression n'est jamais négative). On voit facilement que la quantité $n-m$ ne peut pas augmenter, mais il n'est pas évident qu'elle diminue effectivement chaque fois que le corps de la boucle est exécuté. La valeur du variant ne peut diminuer que via l'une des trois instructions conditionnelles, mutuellement exclusives, à la fin du corps de la boucle. Pour la troisième instruction, la décroissance est évidente : le variant passe d'une valeur strictement positive à la valeur nulle. Pour les deux premières instructions, on constate que les assertions précédant immédiatement les affectations $m := i$ et $n := j$ sont trop faibles : les inégalités devraient être transformées en inégalités strictes. On peut prouver que ce renforcement est possible, mais il n'est pas immédiat et ne sera pas développé ici.

3.6.2.3. *Conclusion*

La première difficulté inhérente à l'approche formelle est le choix des invariants de boucles. Plus ils sont forts, plus ils permettront de démontrer de propriétés. Ceux que nous avons donnés pour le programme "Find" ont permis de démontrer certaines propriétés, mais pas toutes. On observe aussi que la preuve de terminaison repose en bonne partie sur des propriétés relevant de la correction partielle. Pour cette raison, on s'intéressera toujours à la correction partielle avant de considérer la terminaison. Cette dernière propriété impliquera parfois un renforcement des propriétés d'invariance déjà démontrées. Signalons néanmoins qu'en général, la longueur globale des conditions de vérification est du même ordre que la longueur du programme. Le programme "Find" se distingue négativement à cet égard ... ce qui le rend délicat à traiter.

La seconde difficulté est liée à la validation des conditions de vérification. Le travail n'est généralement pas difficile, mais il est long. Ce point est approfondi dans le paragraphe suivant.

3.6.3. *Vérification assistée par ordinateur*

Les preuves que nous avons données des conditions de vérification sont très simples (malgré leur nombre et leur longueur relative) et ne requièrent qu'une connaissance élémentaire de la logique.

Puisque la construction et la preuve des conditions de vérification est un travail relativement simple mais fastidieux, il est assez naturel de se demander

si cette tâche ne pourrait pas être confiée à l'ordinateur. Le problème principal est qu'une preuve "humaine" comporte un grand nombre de "raccourcis", le plus souvent inconscients, dont un automate aveugle ne pourrait s'accommoder. Un pas élémentaire pour l'esprit humain sera décomposé en une séquence de pas pour un démonstrateur de théorèmes. A titre d'exemple, nous allons détailler la preuve de la propriété 7 jusqu'au stade où elle peut être traitée par un programme réel de démonstration de théorèmes. On considère successivement les quatre étapes de la preuve.

La première étape consiste à prouver la formule $(H \Rightarrow C)$ où H est

$$\begin{aligned} & m \leq i \wedge j < i \leq f \wedge j \leq n \wedge 1 \leq m \leq f \leq n \leq N \\ \wedge \quad & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge \quad & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq d \leq A(q))]\,, \end{aligned}$$

tandis que C est

$$\begin{aligned} & m \leq i \wedge j < i \wedge 1 \leq i \leq f \leq n \leq N \\ \wedge \quad & \forall p,q,r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge \quad & \forall p,q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq A(q))]\,. \end{aligned}$$

Nous convertissons d'abord H et C en ensembles conjonctifs d'assertions. Les abréviations sont supprimées : $a < b < c$ est remplacé par $(a < b) \wedge (b < c)$. On obtient pour H l'ensemble

$$\begin{aligned} H_1 \quad & m \leq i\,, \\ H_2 \quad & j < i\,, \\ H_3 \quad & i \leq f\,, \\ H_4 \quad & j \leq n\,, \\ H_5 \quad & 1 \leq m\,, \\ H_6 \quad & m \leq f\,, \\ H_7 \quad & f \leq n\,, \\ H_8 \quad & n \leq N\,, \\ H_9 \quad & \forall p,q,r\,[(1 \leq p \wedge p < m \wedge m \leq r \wedge r \leq n \wedge n < q \wedge q \leq N) \\ & \qquad\qquad \Rightarrow (A(p) \leq A(r) \wedge A(r) \leq A(q))]\,, \\ H_{10} \quad & \forall p,q\,[(m \leq p \wedge p < i \wedge j < q \wedge q \leq n) \Rightarrow (A(p) \leq d \wedge d \leq A(q))]\,, \end{aligned}$$

tandis que C devient

$$\begin{aligned} C_1 \quad & m \leq i\,, \\ C_2 \quad & j < i\,, \\ C_3 \quad & 1 \leq i\,, \\ C_4 \quad & i \leq f\,, \\ C_5 \quad & f \leq n\,, \\ C_6 \quad & n \leq N\,, \\ C_7 \quad & \forall p,q,r\,[(1 \leq p \wedge p < m \wedge m \leq r \wedge r \leq n \wedge n < q \wedge q \leq N) \\ & \qquad \Rightarrow (A(p) \leq A(r) \wedge A(r) \leq A(q))]\,, \\ C_8 \quad & \forall p,q\,[(m \leq p \wedge p < i \wedge j < q \wedge q \leq n) \Rightarrow (A(p) \leq A(q))]\,. \end{aligned}$$

Pour chaque terme C_j, il faut trouver un sous-ensemble E_j de $\{1,\ldots,10\}$ tel que l'implication

$$\left(\bigwedge_{i\in E_j} H_i\right) \Rightarrow C_j$$

puisse être démontrée. Bien souvent, ce sous-ensemble comportera un seul élément. L'élimination des hypothèses inutiles est logiquement superflue, mais cruciale au point de vue de l'efficacité.
Voici la liste des implications résultantes:

1. $H_1 \Rightarrow C_1$,
2. $H_2 \Rightarrow C_2$,
3. $(H_1 \wedge H_5) \Rightarrow C_3$,
4. $H_3 \Rightarrow C_4$,
5. $H_7 \Rightarrow C_5$,
6. $H_8 \Rightarrow C_6$,
7. $H_9 \Rightarrow C_7$,
8. $H_{10} \Rightarrow C_8$.

Les cas $1, 2, 4, 5$ et 6 sont élémentaires: l'hypothèse et la conclusion sont identiques. C'est également vrai pour la formule 7, même si ce fait pourrait ne pas être détecté par un démonstrateur automatique; en effet, les formules longues sont scindées avant tout traitement sémantique. L'implication 3 ne requiert que l'application d'une propriété de base de la relation d'ordre. Finalement, seule la dernière implication va réclamer une analyse plus poussée.

Les formules H_{10} et C_8 partagent la même structure de quantification: $\forall p, q\,[\ldots]$. Soient H' et C' les matrices de H_{10} et C_8 respectivement. Il suffit d'établir la validité de l'implication non quantifiée $H' \Rightarrow C'$. Les formules H' et C' sont elles-mêmes des implications, et leur antécédent est commun. Soient H'' et C'' les conséquents respectifs. Il suffit alors de prouver l'implication $H'' \Rightarrow C''$, c'est-à-dire la formule

$$[A(p) \leq d \wedge d \leq A(q)] \Rightarrow [A(p) \leq A(q)]\,,$$

qui découle immédiatement du fait que $\leq$ est une relation d'ordre.

On passe alors à la deuxième étape de la preuve. Il faut démontrer que la formule

$$\begin{array}{ll} & m \leq i \wedge j < i \wedge 1 \leq i \leq f \leq n \leq N \\ \wedge & \forall p, q, r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p, q\,[(m \leq p < i \wedge j < q \leq n) \Rightarrow (A(p) \leq A(q))] \end{array}$$

implique la formule

$$\begin{array}{ll} & m \leq i \wedge j < i \wedge 1 \leq i \leq f \leq n \leq N \\ \wedge & \forall p, q, r\,[(1 \leq p < m \leq r \leq n < q \leq N) \Rightarrow (A(p) \leq A(r) \leq A(q))] \\ \wedge & \forall p, q\,[(m \leq p < i \wedge i \leq q \leq n) \Rightarrow (A(p) \leq A(q))]\,. \end{array}$$

Comme pour la première étape, on commence par récrire hypothèse et conclusion sous forme conjonctive. Le seul terme critique de la conclusion est alors

$$C: \quad \forall p, q\,[(m \leq p \wedge p < i \wedge i \leq q \wedge q \leq n) \Rightarrow (A(p) \leq A(q))]\,.$$

Les termes utiles de l'ensemble correspondant à l'hypothèse sont

$$H_1: \quad j < i\,,$$
$$H_2: \quad \forall p, q\,[(m \leq p \wedge p < i \wedge j < q \wedge q \leq n) \Rightarrow (A(p) \leq A(q))]\,.$$

On doit donc prouver

$$(H_1 \wedge H_2) \Rightarrow C$$

ou, avec les notations introduites pour la première étape,

$$(H_1 \wedge H_2') \Rightarrow C'$$

avec

$$H_1: \quad j < i\,,$$
$$H_2': \quad (m \leq p \wedge p < i \wedge j < q \wedge q \leq n) \Rightarrow (A(p) \leq A(q))\,,$$
$$C': \quad (m \leq p \wedge p < i \wedge i \leq q \wedge q \leq n) \Rightarrow (A(p) \leq A(q))\,.$$

Cette simplification est autorisée parce que les variables p et q n'apparaissent pas dans H_1.

L'hypothèse H_2' et la conclusion C' partagent le même conséquent. On sait que pour prouver $(E \wedge (A \Rightarrow C)) \Rightarrow (B \Rightarrow C)$, il suffit de prouver $(E \wedge B) \Rightarrow A$, c'est-à-dire, dans le cas présent,
$(m \leq p \wedge p < i \wedge i \leq q \wedge q \leq n \wedge j < i) \Rightarrow (m \leq p \wedge p < i \wedge j < q \wedge q \leq n)$.
Cette implication est démontrée par la technique habituelle; l'antécédent et le conséquent sont transformés en ensembles conjonctifs d'assertions. Chaque assertion de la conclusion apparaît aussi comme assertion de l'hypothèse, à l'exception du littéral $j < q$. Cependant, ce littéral est la conséquence logique des littéraux $j < i$ et $i \leq q$, qui appartiennent bien à l'hypothèse. Cela termine la deuxième étape.

La troisième étape fait simplement intervenir un renommage. D'un point de vue logique, cette opération peut être vue comme une instantiation suivie d'une généralisation. De $\forall x\, P(x)$ on déduit $P(y)$, de $P(y)$ on déduit $\forall y\, P(y)$.

La quatrième et dernière étape comporte une preuve non triviale. Le point critique consiste à inférer C de H_1, H_2 et H_3, avec

$$H_1: \quad m \leq i\,,$$
$$H_2: \quad \forall p, q, r\,[(1 \leq p \wedge p < m \wedge m \leq r \wedge r \leq n \wedge n < q \wedge q \leq N) \Rightarrow (A(p) \leq A(r) \wedge A(r) \leq A(q))]\,,$$
$$H_3: \quad \forall p, q\,[(m \leq p \wedge p < i \wedge i \leq q \wedge q \leq n) \Rightarrow (A(p) \leq A(q))]\,,$$
$$C: \quad \forall p, q, r\,[(1 \leq p \wedge p < i \wedge i \leq r \wedge r \leq n \wedge n < q \wedge q \leq N) \Rightarrow (A(p) \leq A(r) \wedge A(r) \leq A(q))]\,.$$

Le détail est laissé au lecteur.

3.6.4. *Conclusion*

L'emploi de la logique de Hoare permet d'obtenir des preuves formelles de la conformité des algorithmes à leurs spécifications. Comme toutes les preuves formelles, les preuves de ce type sont assez longues. Il convient cependant de remarquer que la logique de Hoare permet de séparer la partie purement algorithmique d'une application informatique de la partie classique, liée au domaine des données. En outre, contrairement à une opinion courante, l'approche formelle de la programmation ne s'oppose pas à l'intuition et à la créativité du programmeur. Au contraire, la logique permet au programmeur d'exprimer ses idées dans un langage simple et précis, puis de vérifier leur validité.

3.7. Forme canonique des programmes impératifs

3.7.1. *Introduction*

Nous avons présenté dans ce chapitre deux langages assez différents pour l'expression des programmes séquentiels. Le formalisme SFS a l'avantage d'être très simple : un programme est simplement un ensemble de transitions. Le formalisme structuré, lui, se prête mieux à la conception et à l'analyse des programmes. Néanmoins, ces deux formalismes, ainsi que les dizaines d'autres communément employés, ont la même "puissance" : tout algorithme qui peut se formaliser dans un langage peut aussi se formaliser dans les autres. La formalisation pourra être plus claire et/ou plus concise dans tel formalisme mais, fondamentalement, l'algorithmique est indépendante du choix d'un langage particulier. Avant même les débuts réels de l'informatique, on a conjecturé (thèse de Church) que la puissance d'expression des langages algorithmiques primitifs ne serait pas augmentée par l'adjonction de nouveaux mécanismes.[18] Pour étudier cette question de manière plus précise, il faudrait définir formellement la notion d'algorithme ou de fonction calculable, ce qui semble difficile. Une autre voie, plus concrète, consiste à montrer que tout programme structuré est équivalent à un système formel séquentiel et

[18] On peut artificiellement diminuer la puissance d'expression d'un langage, ce qui est rarement intéressant. L'analyse des programmes structurés est grandement facilitée si on interdit les boucles, mais la portée pratique d'un langage ainsi appauvri est faible. D'autre part, on peut aussi *concevoir* des mécanismes, parfois appelés *oracles*, qui augmentent la puissance du langage. Si on disposait, par exemple, d'un oracle déterminant en un temps fini la validité ou la non-validité d'une formule logico-arithmétique quelconque, la puissance de l'algorithmique serait significativement augmentée. Le problème est que ces oracles ne sont pas *réalisables* par des machines. Signalons aussi que les perfectionnements apportés régulièrement aux ordinateurs augmentent la vitesse à laquelle les calculs sont effectués mais n'enrichissent pas l'ensemble des fonctions théoriquement calculables. En conclusion, les langages habituels représentent sans doute le seul compromis intéressant entre "puissance d'expression" et "réalisabilité".

réciproquement. C'est ce que nous allons faire, après avoir introduit le non-déterminisme dans le formalisme structuré.

3.7.2. *Le non-déterminisme*

Le système formel représenté à la figure 3.6 calcule le plus grand commun diviseur de trois nombres entiers positifs. Il est *non déterministe*, ce qui signifie qu'un état a plusieurs successeurs potentiels. Par exemple, l'état caractérisé par l'égalité $(x, y, z) = (12, 6, 4)$ admet deux successeurs, respectivement caractérisés par les égalités $(x, y, z) = (6, 6, 4)$ et $(x, y, z) = (12, 2, 4)$. Dans le cas présent, cette indétermination n'est pas importante parce que, quel que soit le successeur choisi, l'état final de l'exécution sera caractérisé par l'égalité $(x, y, z) = (2, 2, 2)$. Par contre, le premier système (simpliste) représenté à la figure 3.13 est fondamentalement non déterministe; au départ d'un état vérifiant *at* ℓ, il admet deux exécutions bien distinctes.

Observons que, pour un état initial fixé, un programme non déterministe est susceptible d'admettre une infinité d'exécutions. C'est le cas par exemple du second programme représenté à la figure 3.13. Au départ de l'état σ_0 caractérisé par la formule *at* $\ell \wedge x = 0$, il existe pour tout entier naturel n une exécution de longueur $n+1$ dont l'état final vérifie *at* $m \wedge x = n$; il existe aussi une exécution infinie dont tous les états vérifient *at* ℓ. Néanmoins, le non-déterminisme est *finitaire*: chaque état admet un nombre fini de successeurs potentiels.

$$
\begin{array}{ll}
P = \{\ell, m\}\,, & P = \{\ell, m\}\,, \\
\mathcal{M} = \{x : \text{entier}\}\,, & \mathcal{M} = \{x : \text{entier}\}\,, \\
\mathcal{T} = \{\,(\ell,\ x := 0,\ m)\,, & \mathcal{T} = \{\,(\ell,\ x := x + 1,\ \ell)\,, \\
\qquad (\ell,\ x := 1,\ m)\,\}\,. & \qquad (\ell,\ \mathit{skip},\ m)\,\}\,.
\end{array}
$$

Figure 3.13 *Deux systèmes non déterministes*

On vérifie immédiatement que les programmes structurés sont toujours déterministes (sauf si les affectations elles-mêmes sont non déterministes, ce qui est généralement exclu). On pourrait penser que cette caractéristique rend la programmation structurée moins puissante que le formalisme SFS, mais ce n'est pas le cas. Etant donné un système formel séquentiel non déterministe S et un état initial σ_0, on peut concevoir un programme séquentiel déterministe (structuré ou non) qui construise pas à pas, et niveau par niveau, l'*arbre des exécutions* relatif à S et σ_0. Les nœuds de cet arbre sont des états (caractérisés par des correspondances variables-valeurs); la racine est σ_0 et les descendants directs d'un état ρ sont les successeurs potentiels de ρ dans les exécutions de S. Le non-déterminisme étant finitaire, l'arbre le sera aussi: même si l'arbre lui-même est infini, chaque niveau sera fini.

Quoique le non-déterminisme (finitaire) ne soit pas essentiel,[19] il est quand même très commode, et un algorithme non déterministe est souvent plus concis et plus facile à comprendre que sa version déterministe; il est de loin plus naturel dans le cas assez fréquent où le résultat souhaité se déduit de l'état final de n'importe quelle exécution. Pour ces raisons, il est utile d'introduire le non-déterminisme en programmation structurée. Cela se fait en généralisant la construction "if-then-else". Si $B_1, \ldots, B_n$ sont des conditions et si $S_1, \ldots, S_n$ sont des programmes, alors

$$\begin{array}{lll} IF & B_1 & \longrightarrow S_1 \\ \square & B_2 & \longrightarrow S_2 \\ \vdots & \vdots & \vdots \\ \square & B_n & \longrightarrow S_n \\ FI & & \end{array}$$

est un programme. Etant donné un état initial σ_0, l'état final σ_1 est déterminé comme suit. Tout d'abord, si σ_0 ne vérifie aucune des conditions B_i, il n'y a pas d'état final (l'exécution est "bloquée").[20] Sinon, l'une des conditions B_i vérifiée par σ_0 est sélectionnée arbitrairement et l'instruction correspondante S_i est exécutée; σ_1 est donc le S_i-successeur de σ_0 (ou l'un des successeurs, si S_i est lui-même non déterministe).

Cette construction généralise l'alternative classique, puisque

if B then S_1 else S_2

se récrit

$$\begin{array}{lll} IF & B & \longrightarrow S_1 \\ \square & \neg B & \longrightarrow S_2 \\ FI\,. & & \end{array}$$

On introduit aussi la construction

$$\begin{array}{lll} DO & B_1 & \longrightarrow S_1 \\ \square & B_2 & \longrightarrow S_2 \\ \vdots & \vdots & \vdots \\ \square & B_n & \longrightarrow S_n \\ OD\,, & & \end{array}$$

par définition équivalente au programme

[19] Le non-déterminisme *infinitaire*, ou *essentiel*, autorise un état à avoir une infinité de successeurs. Un tel mécanisme n'est pas réalisable en pratique. Par exemple, on peut *concevoir* une affectation $x := ?$ qui affecte à x un entier naturel arbitraire, mais on ne peut pas la *réaliser*. Le second programme de la figure 3.13 n'en est qu'une approximation, dont la terminaison n'est pas garantie.

[20] Pour éviter ce phénomène d'échec, il suffit d'imposer que la disjonction $BB =_{def} \bigvee_i B_i$ soit valide.

while BB do
$$\begin{array}{lll} IF & B_1 & \longrightarrow S_1 \\ \Box & B_2 & \longrightarrow S_2 \\ \vdots & \vdots & \vdots \\ \Box & B_n & \longrightarrow S_n \\ FI\,, & & \end{array}$$

où BB est la disjonction $\bigvee_{i=1}^{n} B_i$ des gardes. L'exécution progresse tant que l'une au moins des gardes est vraie. Elle s'arrête dès qu'un état falsifiant toutes les gardes est atteint. De même que pour la boucle classique, l'exécution de la boucle non déterministe peut s'arrêter immédiatement (l'état final est égal à l'état initial σ_0 et falsifie toutes les gardes), ou après un nombre fini d'étapes, ou encore jamais (une séquence infinie (σ_n) d'états est générée, qui vérifient tous la disjonction BB).

Remarque. Alors que l'alternative vide $IF\ FI$ suscite un échec, de même que l'alternative IF *false* $\longrightarrow$ *skip* FI, la boucle vide $DO\ OD$ se réduit à *skip*, de même que la boucle DO *false* $\longrightarrow$ *skip* OD.

La construction IF et la construction DO permettent de définir la *programmation structurée*, ou encore le *langage des commandes gardées.* Les règles de formation sont les suivantes :

- Une affectation est un programme.
- Si B est une condition et si S est un programme, alors $B \longrightarrow S$ est une commande gardée.
- Si S_1 et S_2 sont des programmes, alors $S_1; S_2$ est un programme.
- Si $C_1, \ldots, C_n$ sont des commandes gardées, alors $IF\ C_1 \Box \cdots \Box C_n\ FI$ (alternative) et $DO\ C_1 \Box \cdots \Box C_n\ OD$ (itération) sont des programmes.

Remarques. Une commande gardée C n'est pas un programme mais, par abus de langage, on parle parfois du "programme" C plutôt que du programme $IF\ C\ FI$.

Il n'y a pas lieu de distinguer $(S_1; S_2); S_3$ et $S_1; (S_2; S_3)$, qui ont même sémantique (associativité de la composition séquentielle). On omettra donc les parenthèses.

Rappelons que l'alternative vide et l'itération vide sont des programmes licites; $IF\ FI$ est aussi écrit *fail* ou *abort*, tandis que $DO\ OD$ équivaut à l'affectation sans effet *skip*.

On écrit parfois $\Box_{i=1}^{n} C_i$ au lieu de $C_1 \Box \cdots \Box C_n$.

L'extension de la logique de Hoare au langage des commandes gardées est très simple. La règle d'inférence relative à l'alternative devient

$$\frac{\bigwedge_i [\{A \wedge B_i\}\ S_i\ \{C\}]}{\{A\}\ IF\ \Box_i (B_i \longrightarrow S_i)\ FI\ \{C\}}$$

Celle relative à l'itération devient

$$\frac{\bigwedge_i [\{I \wedge B_i\}\ S_i\ \{I\}]}{\{I\}\ DO\ \Box_i(B_i \longrightarrow S_i)\ OD\ \{I \wedge \bigwedge_i \neg B_i\}}$$

Remarque. Le langage des commandes gardées est essentiellement utilisé pour le développement de programmes qui, moyennant la satisfaction de conditions initiales spécifiées, se terminent toujours. Dans ce contexte, on utilisera le calcul *wp* mentionné au paragraphe 3.2.8 plutôt que la logique de Hoare ou le calcul *wlp-slp*.

3.7.3. *Techniques de conversion*

La conversion d'un système formel séquentiel en programme structuré est fort simple. Soit $\mathcal{S} = (P, \mathcal{M}, \mathcal{T})$ un tel système. Le programme structuré équivalent sera une itération comportant autant de commandes gardées que $\mathcal{S}$ comporte de transitions. A la transition $(\ell, B \longrightarrow x := e, m)$ correspondra la commande gardée $(Z = \ell \wedge B \longrightarrow (x, Z) := (e, m))$, où Z est une nouvelle variable dont le type est l'ensemble P des étiquettes de $\mathcal{S}$. Par exemple, le système séquentiel de calcul de la factorielle présenté au paragraphe 3.4.1 sera converti en l'itération

$$\begin{aligned}
&DO \quad Z = \ell_0 \longrightarrow (y, Z) := (1, \ell_1)\\
&\Box \quad Z = \ell_1 \wedge x > 0 \longrightarrow (y, Z) := (y * x, \ell_2)\\
&\Box \quad Z = \ell_2 \longrightarrow (x, Z) := (x - 1, \ell_1)\\
&\Box \quad Z = \ell_1 \wedge x = 0 \longrightarrow (F, Z) := (y, \ell_3)\\
&OD\,.
\end{aligned}$$

Cette technique de conversion est simple et générale, mais n'introduit pas de structure. Si I est un invariant du système $\mathcal{S}$, un invariant de l'itération s'obtient en remplaçant dans I toute occurrence de *at* ℓ par $Z = \ell$, pour tout $\ell \in P$. La structuration permet souvent d'éviter l'introduction de la variable supplémentaire Z. Dans le cas présent, un programme structuré plus élégant serait

$$\begin{aligned}
&y := 1;\\
&DO\ x > 0 \longrightarrow y := y * x;\ x := x - 1\ OD;\\
&F := y.
\end{aligned}$$

Les règles formelles permettant la conversion inverse sont plus lourdes, mais le processus reste fort simple. L'idée est d'utiliser les étiquettes pour imposer l'ordonnancement qui, en programmation structurée, est induit par l'opérateur de composition séquentielle. Nous donnons à titre d'exemple le résultat de la conversion pour l'algorithme de tri par insertion, dont une version en programmation structurée est

$i := 1;$
$DO\ i \leq n \longrightarrow x := A[i];$
$\qquad j := i;$
$\qquad DO\ j > 1 \wedge A[j-1] > x \longrightarrow A[j] := A[j-1];$
$\qquad\qquad j := j - 1$
$\qquad OD;$
$\qquad A[j] := x;$
$\qquad i := i + 1$
$OD.$

Un système séquentiel équivalent comportera les transitions suivantes :

$(\ell_0,\ i := 1\ ,\ \ell_1)\ ,$
$(\ell_1,\ i \leq n \longrightarrow x := A[i]\ ,\ \ell_2)\ ,$
$(\ell_2,\ j := i\ ,\ \ell_3)\ ,$
$(\ell_3,\ j > 1 \wedge A[j-1] > x \longrightarrow A[j] := A[j-1]\ ,\ \ell_4)\ ,$
$(\ell_4,\ j := j - 1\ ,\ \ell_3)\ ,$
$(\ell_3,\ \neg(j > 1 \wedge A[j-1] > x) \longrightarrow A[j] := x\ ,\ \ell_5)\ ,$
$(\ell_5,\ i := i + 1\ ,\ \ell_1)\ ,$
$(\ell_1,\ i > n \longrightarrow skip\ ,\ \ell_6)\ .$

On a $P = \{\ell_0, \ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}$; un état initial doit satisfaire *at* ℓ_0 et un état final doit satisfaire *at* ℓ_6.

3.7.4. *Affectations, conditions et procédures*

Les techniques de conversion que nous venons de voir présupposent que les composants élémentaires des programmes, c'est-à-dire les constantes (données de base), les affectations et les conditions (ou gardes), étaient les mêmes dans tous les langages de programmation. En fait, dans les paragraphes 3.2.1 et suivants, nous avons insisté sur les propriétés sémantiques des objets de base, mais nous n'en avons pas donné de liste exhaustive. Il se fait que la plupart des langages de programmation usuels offrent assez peu d'objets élémentaires mais permettent à l'utilisateur d'en construire de nouveaux, au moyen d'un mécanisme que nous allons maintenant introduire.

Beaucoup de langages ne permettent pas l'affectation $x := a^b$ où x est une variable numérique et où a et b sont des expressions numériques. Pour simplifier, nous supposons que les valeurs possibles de x et a sont des entiers, et celles de b sont des entiers naturels. A défaut de l'affectation $x := a^b$, on peut utiliser le programme suivant :

$u := a;$
$v := b;$
$x := 1;$
while $v > 0$ do
$\qquad x := x * u;$
$\qquad v := v - 1.$

Une variante, plus efficace quand b est grand, est

$$
\begin{array}{l}
u := a; \\
v := b; \\
x := 1; \\
DO \quad v > 0 \wedge \mathrm{pair}(v) \longrightarrow u := u * u; v := v/2 \\
\quad \square \quad v > 0 \wedge \mathrm{impair}(v) \longrightarrow x := x * u; v := v - 1 \\
OD\,.
\end{array}
$$

Dans ces programmes, u et v désignent des nouvelles variables, n'apparaissant pas dans les expressions a et b. De plus, si ces programmes sont des composants de programmes plus grands, ceux-ci ne peuvent pas non plus évoquer ces variables. Les boucles de ces programmes admettent l'invariant

$$(0 \leq v \leq b) \wedge (x * u^v = a^b)\,.$$

Dans le premier programme, u est constant et égal à a; l'existence de u est théoriquement superflue, mais améliore l'efficacité si a est une expression dont l'évaluation est compliquée.

Le mécanisme de procédure consiste essentiellement à donner un nom à l'un de ces programmes, par exemple $power(x, a, b)$. Une fois ce nom donné, un utilisateur pourra disposer de l'affectation correspondante, et il écrira par exemple $power(y, c, d)$ pour obtenir exactement l'effet de l'affectation (illicite et fictive) $y := c^d$. Un programme ainsi doté d'un nom, et utilisable comme affectation, est une *procédure*.

Le mécanisme procédural a aussi d'autres effets. Notamment, il permet à l'utilisateur d'ignorer (sous certaines conditions) le nom des *variables locales*, ici u et v. Si donc l'utilisateur introduit, dans un programme utilisant la procédure *power*, des variables u et v, celles-ci seront considérées comme distinctes des variables locales de la procédure.

Dans le nom $power(x, a, b)$ attribué à la procédure, x, a et b sont des *paramètres formels*. Le mot "formel" signifie que ces variables servent à la définition de la procédure mais n'existent pas nécessairement dans un programme *utilisant* la procédure. (Un tel programme peut contenir par hasard certaines de ces variables.) Quand la procédure est utilisée, par exemple au moyen de l'écriture $power(y, c, d)$, les variable et expressions y, c et d sont les *paramètres réels*. Le mot "réel" souligne que les paramètres ont une existence réelle dans le programme contenant l'*appel* "$power(y, c, d)$" à la procédure *power*. Le mécanisme procédural vérifie aussi la *compatibilité* entre les paramètres réels et les paramètres formels correspondants, pour chaque appel. En particulier, les types des paramètres correspondants doivent être les mêmes. Notons enfin que le paramètre formel x est un paramètre *résultat* tandis que les paramètres a et b sont des paramètres *valeurs*, ou *données*. Cela signifie que, dans le triplet de Hoare spécifiant la procédure, x apparaîtra dans

la postcondition, tandis que a et b apparaîtront dans la précondition.[21] Ce triplet sera

$$\{a = a_0 \in \mathbf{Z} \wedge b = b_0 \in \mathbf{N}\}\ power(x, a, b)\ \{x = a_0^{b_0}\}\,.$$

C'est au concepteur d'une procédure qu'il appartient de fixer le caractère valeur ou résultat d'un paramètre. Il écrira, par exemple, *power*(res x, val a, val b). Cette précision permettra d'accepter, dans la spécification de la procédure, l'abus d'écriture suivant :

$$\{a \in \mathbf{Z} \wedge b \in \mathbf{N}\}\ power(x, a, b)\ \{x = a^b\}\,.$$

Les caractères valeur et résultat ne sont pas incompatibles. On peut fort bien définir une procédure *sort*(A) de tri *in situ* acceptant comme paramètre "valeur-résultat" un tableau A. L'exécution de l'appel *sort*(B) aura pour effet de trier le tableau B.

Voici à titre d'exemple un programme spécifié contenant un appel à la procédure *power*.

$\{\forall i\,(1 \leq i \leq n \Rightarrow (A[i] \in \mathbf{Z} \wedge B[i] \in \mathbf{N}))\}$
$s := 0;$
$j := 0;$
while $j < n$ do
    $j := j + 1;$
    $power(r, A[j], B[j]);$
    $s := s + r$
$\{s = \sum_{i=1}^{n} A[i]^{B[i]}\}\,.$

L'invariant de boucle est

$$0 \leq j \leq n \wedge s = \sum_{i=1}^{j} A[i]^{B[i]}\,.$$

Conceptuellement, le mécanisme procédural est assez simple. Il s'agit essentiellement d'une technique d'abstraction paramétrée, permettant au programmeur d'enrichir le langage qu'il utilise par de nouvelles affectations (qui peuvent être conditionnelles et/ou non déterministes). En informatique, la conception détaillée et la mise en œuvre du mécanisme procédural constituent un problème essentiel et souvent difficile. Sans aborder ce problème ici, soulignons que le mécanisme procédural peut être étendu de plusieurs manières. Nous aurons l'occasion d'y revenir au chapitre 5.

[21] Les variables locales n'apparaissent pas dans la spécification.

Chapitre 4
Logique pour programmes parallèles

4.1. Introduction

En programmation impérative, l'approche séquentielle présuppose qu'une tâche unique est accomplie par une seule machine. S'il faut accomplir plusieurs tâches, un ordre strict sera imposé entre elles, et une tâche ne pourra être entamée que si la tâche précédente est achevée. Si plusieurs machines sont disponibles, elles ne pourront être utilisées que pour des tâches distinctes.

En pratique, on est souvent confronté à des situations où les restrictions inhérentes au paradigme séquentiel sont très gênantes. Un premier exemple est celui d'une tâche qui doit être accomplie à une vitesse supérieure à celle que permet l'usage d'une machine isolée; on peut alors se demander si plusieurs machines, interconnectées de manière adéquate, ne pourraient pas concourir à l'accomplissement de la tâche à une vitesse suffisante. En particulier, dans le cas de problèmes mettant en jeu de très grandes structures de données (tri d'un grand tableau, résolution d'un grand système d'équations linéaires, etc.), on voudrait pouvoir utiliser un "tableau de machines" dont chaque élément ne traiterait que peu de données à la fois.

Un deuxième cas fréquent est celui où une seule machine est partagée entre plusieurs utilisateurs. Si la machine est suffisamment puissante, il est possible de la programmer en sorte que chaque utilisateur ait l'impression d'en disposer seul. C'est l'approche "temps partagé", où les tâches de tous les utilisateurs semblent progresser en même temps.

Un troisième cas est celui de tâches qui, par leur nature même, échappent à l'approche séquentielle. Un cas typique est celui des réservations des places d'avion. On peut réserver une place sur un vol donné à partir des ordinateurs de

dizaines d'agences de voyage, réparties sur un vaste territoire. Ces ordinateurs concourent visiblement à une même tâche.

L'objectif de la programmation parallèle est d'apporter des solutions fiables aux problèmes pour lesquels l'approche séquentielle est inadaptée. Le parallélisme joue actuellement un rôle crucial dans les systèmes informatiques. Le système d'exploitation, c'est-à-dire le logiciel de base d'un ordinateur, est organisé sous forme d'un ensemble de programmes exécutés en parallèle. Les réseaux d'ordinateurs, composés de plusieurs machines fonctionnant en parallèle, sont de plus en plus répandus et, dans de nombreux cas, procurent un service hors des capacités des machines isolées, si puissantes soient-elles.

La programmation parallèle est cependant plus difficile que la programmation séquentielle. A taille égale, la mise au point d'un programme parallèle est beaucoup plus longue que la mise au point d'un programme séquentiel. La raison est intuitivement simple; le programmeur n'a plus seulement à se soucier de tâches isolées, mais aussi de la façon dont ces tâches peuvent interférer. Une présentation plus formelle des difficultés inhérentes à la programmation parallèle est donnée au paragraphe suivant.

La complexité de la programmation parallèle rend d'autant plus nécessaire une approche formelle mais, contrairement à ce que nous avons vu pour la programmation séquentielle, aucune approche formelle ne fait l'unanimité en ce qui concerne le parallélisme. Plutôt que de passer en revue les nombreux systèmes formels proposés par les chercheurs durant ces dernières années, nous avons choisi de ne présenter que la méthode mise au point par le second auteur, essentiellement dans le cadre du projet Esprit ATES. Sans être intrinsèquement supérieure aux approches alternatives, cette méthode a l'avantage d'être simple et accessible aux débutants. Elle ne nécessite l'introduction d'aucun nouveau concept mathématique et préserve la plupart des qualités des méthodes qui ont fait leurs preuves en programmation séquentielle.

4.2. Parallélisme et structuration

Dans la mesure où la logique de Hoare est utile en programmation séquentielle, il est naturel de chercher à l'étendre à la programmation parallèle. Le moyen le plus élémentaire d'introduire le parallélisme est, à première vue, l'introduction d'un mécanisme de programmation supplémentaire, dénoté "$\|$", tel que $(S\|R)$ dénote la composition parallèle des programmes S et R. On peut donner plusieurs significations à l'opérateur $\|$, mais la plus simple est la suivante.

> Une exécution de $(S\|R)$ est obtenue en entrelaçant une exécution de S et une exécution de R. Plus précisément, un pas d'exécution de $(S\|R)$ consiste soit en un pas d'exécution de S, soit en un pas d'exécution de R. La décision, à chaque pas, de faire progresser S ou R est prise arbitrairement par une entité nommée le *séquenceur*.

Cette "définition" donne une importance excessive à la notion de *pas d'exécution*. Un exemple très simple suffit à expliquer ce problème. Supposons que S et R consistent tous deux en l'unique instruction $x := x + 1$. Il semble qu'il n'existe que deux exécutions possibles de $(S \| R)$, et que ces deux exécutions auront le même effet, décrit par le triplet de Hoare

$$\{x = x_0\}\ S \| R\ \{x = x_0 + 2\}\,.$$

Ce ne sera pourtant pas vrai si une affectation est considérée comme une séquence d'instructions élémentaires plutôt que comme une seule instruction. Si par exemple, l'affectation $x := x + 1$ est remplacée, à l'exécution, par la séquence

$$a := x;\ a := a + 1;\ x := a\,,$$

alors, certaines exécutions de $(S \| R)$ n'auront pas l'effet attendu.[1] L'une d'elles s'obtient en exécutant la séquence

$$a := x;\ a := a + 1;\ a := x;\ a := a + 1;\ x := a;\ x := a\,.$$

En effet, l'exécution de cette séquence augmente le contenu de x d'une unité seulement.

La conclusion est que, dans ce contexte, il convient de définir un concept d'instruction atomique. Une *instruction atomique* est une instruction dont l'exécution s'étend sur un seul pas d'exécution. Pour signifier qu'une instruction est atomique, on l'entoure parfois des parenthèses angulaires $\langle$ et $\rangle$. Par exemple, le triplet de Hoare

$$\{x = x_0\}\ \langle x := x + 1\rangle \| \langle x := x + 1\rangle\ \{x = x_0 + 2\}$$

est valide.

Remarque. Des mécanismes existent, au niveau du matériel comme à celui du système d'exploitation de l'ordinateur, pour imposer le respect, à l'exécution, du caractère atomique d'une instruction. Ces mécanismes permettent de "protéger" l'exécution d'une affectation, mais aussi d'une section plus longue d'un programme.

Notre propos n'est pas ici de décrire les subtilités de l'exécution parallèle des programmes, mais simplement de souligner le fait suivant. Même si, considérés isolément, les programmes S_1 et S_2 ont le même comportement, cela n'implique pas que les programmes $(S_1 \| R)$ et $(S_2 \| R)$ auront aussi le même comportement. D'une manière plus générale, la fonctionnalité de la composition parallèle des programmes S et R ne s'exprime pas en termes de la fonctionnalité de S et de la fonctionnalité de R. On dira que l'opération $\|$ n'est pas *compositionnelle*. Ce phénomène est inhabituel en mathématique, où les opérateurs sont compositionnels. Considérons par exemple l'addition. La somme de deux expressions arithmétiques e_1 et e_2 ne dépend que des valeurs

[1] Un tel remplacement, séquentiellement correct, est parfois effectué par le système d'exploitation.

de ces expressions et non de leur structure interne. C'est d'ailleurs cette propriété qui permet l'évaluation progressive et méthodique des expressions arithmétiques. Pour les connecteurs logiques, compositionnalité est synonyme de vérifonctionnalité.

Il est cependant un cas particulier où la composition parallèle de deux programmes ne comporte pas de difficulté; c'est celui où les deux programmes se réduisent à des affectations évoquant des ensembles disjoints de variables. Dans ce cas, $\langle x := e(x)\rangle \| \langle y := f(y)\rangle$ se réduit tout simplement à $(x, y) := (e(x), f(y))$ et la séquentialisation est possible des deux manières. Un cas moins restrictif est celui de l'affectation $(x, y) := (e(x), f(x, y))$, équivalente à la séquence $y := f(x, y); x := e(x)$, sans l'être à la séquence $x := e(x); y := f(x, y)$. Paradoxalement, ce type élémentaire de parallélisme permet l'élaboration de programmes parallèles non triviaux. Nous donnerons un exemple au paragraphe 4.3.

Dans la plupart des cas, le parallélisme élémentaire est insuffisant, et divers moyens ont été introduits pour pallier les défauts de la composition parallèle. Comme ces moyens sont relativement compliqués, tant du point de vue des fondements théoriques que des applications pratiques, nous ne les évoquerons pas ici.

Une autre approche, plus radicale, consiste à ne pas introduire d'opérateur de composition parallèle. Un système de processus[2] distribués sera décrit comme un tout, et non comme résultant de la composition parallèle de processus individuels. Cette approche a l'avantage d'être simple, mais l'inconvénient d'être non structurée; elle donne néanmoins de bons résultats. Nous l'illustrerons en développant des systèmes distribués dans le formalisme des *systèmes formels concurrents* (§ 4.4). Citons ici quelques contributions importantes à l'extension aux programmes parallèles de techniques utiles en programmation séquentielle: Lipton (1975), Keller (1976), Owicki et Gries (1976), Dijkstra (1978, 1979), Lamport (1980, 1983, 1987), Sifakis (1982), Gerth (1984), Back et Kurki-Suonio (1988), Chandy et Misra (1988).

4.3. Parallélisme implicite : le tri systolique

Nous développons ici une application du "parallélisme implicite". Le développement se fait dans le cadre séquentiel, mais la réalisation concrète de l'application utilise un ensemble de cellules fonctionnant en parallèle.

4.3.1. *Le problème*

Nous avons vu que l'efficacité des programmes de tri élémentaires est quadratique, c'est-à-dire que le temps d'exécution est proportionnel au carré

[2] Dans ce contexte, un *processus* est un programme spécifiquement conçu pour fonctionner en parallèle avec d'autres programmes.

de la taille n du tableau à trier. On peut imaginer que si n machines (ou un nombre proportionnel à n) sont disponibles, on peut les connecter de manière à obtenir une "super-machine" capable de trier en un temps proportionnel à n.

La figure 4.1 présente intuitivement une manière de procéder, sous forme d'une machine cellulaire à $2n + 1$ éléments. Le flux des données est introduit dans la machine par la cellule inférieure gauche, et le flux des résultats (données triées) est extrait de la cellule supérieure gauche.

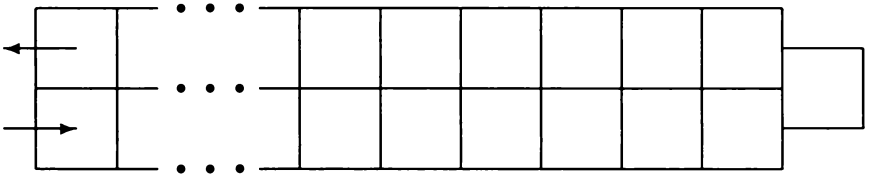

Figure 4.1 *Architecture de tri parallèle d'un flux de données*

La machine peut exécuter trois actions :

1. entrée d'une donnée;
2. sortie d'un résultat;
3. tri binaire entre cellules appariées hautes et basses.

L'opération d'entrée est illustrée à la figure 4.2. Chaque élément "pousse" son voisin vers la droite (ce qui provoque l'effacement de l'élément terminal). Considérons deux cellules contiguës; l'affectation modélisant leur comportement sera du type $(g, d, out) := (in, g, d)$, la cellule g étant à gauche de la cellule d. On est dans le cas où la séquentialisation est possible, mais dans un sens seulement, à savoir $out := d$; $d := g$; $g := in$.

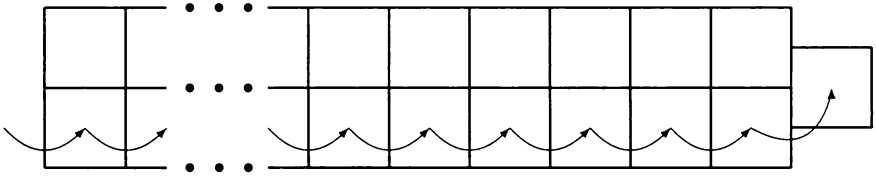

Figure 4.2 *Introduction d'un élément du flux d'entrée*

L'opération de sortie est illustrée à la figure 4.3. Elle est analogue à l'opération d'entrée. Le décalage vers la gauche provoque ici la duplication de l'élément terminal. A nouveau, la séquentialisation est possible, mais dans un seul sens.

L'opération de tri multiple est illustrée à la figure 4.4. Il n'y a pas de communication latérale, ni de restriction à la séquentialisation.

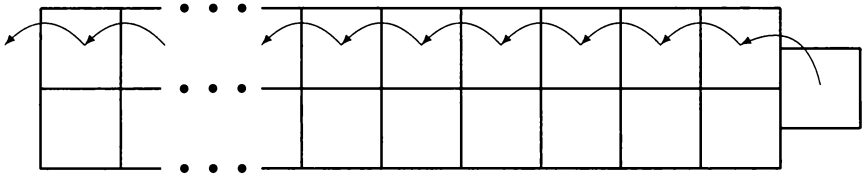

Figure 4.3 *Extraction d'un élément du flux de sortie*

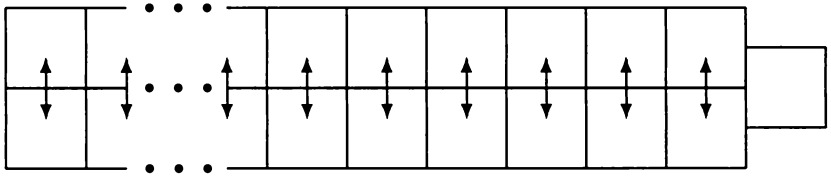

Figure 4.4 *Composition parallèle de n tris binaires indépendants*

L'action d'entrée, notée *IN*, provoque la destruction d'une donnée, tandis que l'action de sortie, notée *OUT*, provoque la duplication d'un élément. Ces deux actions seront donc exécutées en stricte alternance, de sorte que la valeur détruite par *IN* soit toujours la valeur dupliquée lors de la dernière exécution de *OUT*. L'action de tri parallèle est notée *BIN*. Son effet est de faire "remonter" les valeurs les plus "légères" dans les cellules du haut.

Si la machine cellulaire est capable de trier, la plus petite valeur qu'elle contient doit se trouver en haut à gauche, puisque là se trouve la prochaine valeur à extraire. A supposer que cette propriété soit vraie à un instant donné, toute action d'entrée ou de sortie est susceptible de la détruire, et le seul moyen dont nous disposions pour la restaurer est l'exécution de l'action *BIN*. Notons enfin que cette opération est idempotente : les programmes *BIN* et *BIN* ; *BIN* sont équivalents. Il est donc inutile d'exécuter *BIN* deux fois de suite.

Ce raisonnement simple nous permet de déduire que toute exécution pertinente sur la machine cellulaire sera du type

$$\ldots\ IN\ ;\ BIN\ ;\ OUT\ ;\ BIN\ ;\ IN\ ;\ BIN\ ;\ OUT\ ;\ BIN\ ;\ IN\ \ldots,$$

sans que la longueur de la séquence, ni ses premier et dernier éléments, ne soient déterminés à ce stade.

4.3.2. *Le programme de base et son invariant*

Avant de mettre en œuvre concrètement la machine cellulaire, nous allons la modéliser, de même que son cycle de base *IN* ; *BIN* ; *OUT* ; *BIN*.

L'ensemble des cellules est modélisé par le tableau $V[-n : n]$. La cellule supérieure gauche est $V[-n]$, l'inférieure gauche est $V[n]$ et la cellule de

l'extrémité droite est $V[0]$. Cette dernière est isolée tandis que les autres sont groupées en n paires de cellules superposées. Soient $A[1,2,\ldots]$ le flux de données entrant et $B[1,2,\ldots]$ le flux de résultats sortant. On utilise les abréviations suivantes : $*S$ signifie "répéter indéfiniment le programme S", et $*(r)S$ signifie "répéter r fois le programme S". En outre, "$sort(1..n)$" abrège "$sort(1) \parallel \cdots \parallel sort(n)$", et "$sort(i)$" abrège

"if $V[i] < V[-i]$ then $(V[i], V[-i]) := (V[-i], V[i])$".

Le comportement global du système et de son environnement est modélisé par le programme $*[cycle\ de\ base]$ donné à la figure 4.5. Ce programme est la "réduction séquentielle" de processus distribués dans les cellules; ceux-ci seront déterminés plus loin.

$$
\begin{array}{l}
*[\ V[0:n-1] := V[1:n]\,;\ V[n] := A[j]\,; \\
\quad sort(1..n)\,; \\
\quad B[j] := V[-n]\,;\ V[-n:-1] := V[1-n:0]\,; \\
\quad sort(1..n)\,; \\
\quad j := j+1\ ]\,.
\end{array}
$$

Figure 4.5 *Réduction séquentielle générique du système de tri*

Le système cellulaire doit trier en ordre croissant, donc chaque résultat sortant doit être au plus égal à toute valeur contenue dans le système. La prochaine valeur à sortir étant $V[-n]$, l'assertion exprimant cette propriété d'invariance requise du système sera

$$V[-n] \;=\; min(V[-n:n])\,.$$

En outre, chaque segment droit $V[-i:i]$ du système de tri entier $V[-n:n]$ est aussi un système de tri; en conséquence, nous généralisons la propriété en

$$I \;=_{def}\; \forall i\,(\,0 \le i \le n \;\Rightarrow\; V[-i] = min(V[-i:i])\,)\,.$$

Nous devons démontrer la proposition suivante.

Théorème 4.1. Si $in ::= [IN\,;BIN]$ et $out ::= [OUT\,;BIN]$ sont respectivement le programme

$$V[0:n-1] := V[1:n];\ V[n] := A[j];\ sort(1..n)$$

et le programme

$$B[j] := V[-n];\ V[-n:-1] := V[1-n:0];\ sort(1..n)$$

et si I est l'invariant introduit ci-dessus, alors les triplets

$$\{I\}\ in\ \{I\}\ ,\quad \{I\}\ out\ \{I\}$$

sont vrais.

Preuve. Voici un schéma de preuve du premier triplet.

$$\{\forall i\,[0 \le i \le n \;\Rightarrow\; V[-i] = min(V[-i:i])\,]\}$$
$$V[0:n-1] := V[1:n]\,;V[n] := A[j];$$
$$\{\forall i\,[0 < i \le n \;\Rightarrow\; V[-i] = min(V[-i:i-1]) \le min(V[1-i:i-1])]\}$$
$$sort(1..n)$$
$$\{\forall i\,[0 < i \le n \;\Rightarrow\; V[-i] = min(V[-i], V[i]) \le min(V[1-i:i-1])]\}$$
$$\{\forall i\,[0 \le i \le n \;\Rightarrow\; V[-i] = min(V[-i:i])]\}$$

La preuve du second triplet est similaire.

4.3.3. *Comment trier une suite de nombres ?*

Nous venons de démontrer que le système de tri maintenait les valeurs des cellules supérieures triées en ordre croissant. En effet, un corollaire immédiat du théorème 4.1 est la propriété

$$V[-n] \le V[1-n] \le \cdots \le V[-1] \le V[0]\,.$$

Cela n'implique pas en soi que le système trie effectivement le flux d'entrée. Il faut tout d'abord prévoir les valeurs initiales des cellules du système. Comme ces valeurs ne doivent en aucun cas se mélanger au flux d'entrée, il faut qu'elles soient inférieures à chaque entrée (nous désignerons par $-\infty$ une valeur arbitraire mais plus petite que toute entrée possible). Pendant une première phase de l'exécution, le flux d'entrée T est introduit dans le système tandis qu'un flux Y composé d'occurrences de $-\infty$ en est extrait. Pendant la seconde phase, un flux Z composé d'occurrences de $+\infty$ est introduit dans le système (on suppose que $+\infty$ est une valeur supérieure à toute entrée possible) et le flux S des données triées est collecté à la sortie du système, sans se "mélanger" au flux Z, qui reste à l'intérieur du système.

La taille maximale du flux T à trier est naturellement $2n+1$, puisque le système comporte $2n+1$ cellules. Il reste à déterminer la structure de l'exécution permettant ce tri. On connaît à son sujet les points suivants :

- elle se compose d'étapes *in* et *out*, se succédant en stricte alternance;
- elle commence par *in* (un *out* préliminaire serait inutile);
- l'instant intermédiaire de l'exécution où les $2n+1$ valeurs à trier coexistent dans le système est précédé de l'exécution de *in*, et suivi de l'exécution de *out*;
- l'exécution se termine par un *out* (un *in* supplémentaire serait inutile).

La plus courte exécution vérifiant ces critères est *in* ; *out*; on vérifie immédiatement qu'elle correspond au cas $n = 0$ et que la machine cellulaire, réduite à sa cellule terminale, se comporte en simple mémoire tampon. La taille des flux T et S est 1; la taille du flux Y est nulle, car l'étape *in* détruit l'unique valeur initiale; enfin, la taille du flux Z est nulle, car la "valeur à trier" ressort de la machine à l'étape *out*, avant que l'on ait l'occasion d'injecter la première valeur $+\infty$.

On vérifie également, par essai direct, que la séquence

$$in\,;\ out\,;\ in\,;\ out\,;\ in\,;\ out\,;\ in\,;\ out\,;\ in\,;\ out\,,$$

composée de cinq cycles de base, correspond au cas $n = 1$ et permet de trier $2n + 1 = 3$ valeurs. Ces trois valeurs coexistent dans la machine au milieu du troisième cycle. La taille des flux T et S est 3 et la taille des flux Y et Z est 2.

Il est maintenant facile d'induire la structure de l'exécution générale (ce qui ne dispense naturellement pas de la vérification formelle a posteriori). La taille des flux Y et Z est $2n$, celle des flux T et S est $2n + 1$, l'exécution se compose de $4n + 1$ cycles de base et les $2n + 1$ valeurs à trier coexistent dans la machine au milieu du $(2n+1)$ième cycle.

Initialement, le flux d'entrée se compose de $2n + 1$ valeurs à trier formant le tableau $T[-n:n]$, suivies de $2n$ occurrences de $+\infty$. A chaque étape d'entrée, ce flux est amputé de sa valeur de tête, qui entre dans le système. Après $4n+1$ cycles, le flux d'entrée est vide et l'exécution s'arrête. Le flux de sortie évolue en sens inverse. Il est initialement vide et, à chaque étape de sortie, une valeur extraite du système est ajoutée en queue de flux. Dans l'état final, le flux de sortie comporte $4n+1$ valeurs; les $2n$ premières sont égales à $-\infty$, les $2n+1$ dernières forment la version triée $S[-n:n]$ du tableau $T[-n:n]$.

La figure 4.6 décrit l'état du tableau $V[-n : n]$ après un certain nombre de cycles (entier ou demi entier); cette description équivaut à un invariant. Rappelons aussi que l'invariant I introduit au paragraphe précédent est vrai avant et après toute étape *in* ou *out*; il n'est donc pas systématiquement répété. On peut aussi observer qu'après chaque action OUT on a $V[0] = V[-1]$; après chaque étape *out* (nombre entier de cycles), on a donc $V[0] \in \{V[-1], V[1]\}$. Pour tenir compte de cette duplication, la notation V_0 désignera le tableau V privé de sa composante $V[0]$; ce tableau est donc indicé de $-n$ à -1 puis de 1 à n.

Dans la figure 4.6, la première colonne représente le nombre de cycles écoulés et la deuxième colonne décrit le tableau V. Le tableau $S[-n : n]$ dénote la version triée du tableau $T[-n:n]$. La notation $\pi(A, B)$ signifie que le tableau B est une permutation du tableau A; en particulier, on a $\pi(T[-n:n], S[-n:n])$.

Remarque. La machine cellulaire à $2n+1$ composantes peut trier en temps réel un flux illimité à condition qu'il vérifie la propriété suivante : toute valeur entrant dans la machine est au moins égale à toute valeur qui en est déjà sortie. Il n'est pas facile de déterminer par inspection si cette propriété est vraie pour un flux donné, mais il est très simple de la tester en temps réel.

4.3.4. *Réalisation séquentielle*

L'algorithme étudié permet de trier $2n+1$ valeurs en $4n+1$ étapes. Cependant, il faut observer que chaque étape implique des affectations multiples, concernant n ou $2n + 1$ variables. La réalisation de cet algorithme sur une machine séquentielle est représentée à la figure 4.7; chaque affectation multiple

cycle(s)	$V[-n{:}n]$
0	$V[-n{:}n] = \langle -\infty \rangle$
$n-j$	$V[-n{:}j] = \langle -\infty \rangle \wedge V[j{+}1{:}n] = T[-n{:}-j-1]$
$n-j+\frac{1}{2}$	$V[-n{:}j-1] = \langle -\infty \rangle \wedge V[j{:}n] = T[-n{:}-j]$
$n+k$	$V[-n{:}-k-1] = \langle -\infty \rangle \wedge \pi(V_0[-k{:}n], T[-n{:}k-1])$
$n+k+\frac{1}{2}$	$V[-n{:}-k-1] = \langle -\infty \rangle \wedge \pi(V[-k{:}n], T[-n{:}k])$
$2n+\frac{1}{2}$	$\pi(V[-n{:}n], S[-n{:}n]) \wedge \pi(V[-n{:}n], T[-n{:}n])$
$3n+\frac{1}{2}-j$	$\pi(V[-n{:}j], S[-j{:}n]) \wedge V[j{+}1{:}n] = \langle +\infty \rangle$
$3n+1-j$	$\pi(V_0[-n{:}j], S[1{-}j{:}n]) \wedge V[j{+}1{:}n] = \langle +\infty \rangle$
$3n+\frac{1}{2}+k$	$V[-n{:}-k] = S[k{:}n] \wedge V[-k{+}1{:}n] = \langle +\infty \rangle$
$3n+1+k$	$V[-n{:}-k-1] = S[k{+}1{:}n] \wedge V[-k{:}n] = \langle +\infty \rangle$
$4n+1$	$V[-n{:}n] = \langle +\infty \rangle$

Figure 4.6 *Description de l'exécution générale, avec* $0 \leq j \leq n$ *et* $1 \leq k \leq n$

a été remplacée par une séquence d'affectations simples. L'écriture "sort(i)" abrège le programme

$$\text{if } V[i] < V[-i] \text{ then } (V[i], V[-i]) := (V[-i], V[i])\,.$$

$$
\begin{array}{l}
j := 1; \\
*(4n{+}1)\,[\, i := 1\,;\, *(n)\,[\, V[i{-}1] := V[i]\,;\, i := i{+}1\,]\,; \\
\qquad\qquad \text{if } j \leq 2n{+}1 \text{ then } V[n] := T[j{-}n{-}1] \text{ else } V[n] := +\infty\,; \\
\qquad\qquad i := 1\,;\, *(n)\,[\,\text{sort}(i)\,;\, i := i+1\,]\,; \\
\qquad\qquad \text{if } j \geq 2n{+}1 \text{ then } S[j{-}3n{-}1] := V[-n]\,; \\
\qquad\qquad i := -n\,;\, *(n)\,[\, V[i] := V[i{+}1]\,;\, i := i{+}1\,]\,; \\
\qquad\qquad i := 1\,;\, *(n)\,[\,\text{sort}(i)\,;\, i := i+1\,]\,; \\
\qquad\qquad j := j+1\,]\,.
\end{array}
$$

Figure 4.7 *Réalisation séquentielle — et inefficace — du système de tri*

4.3.5. *Communication synchrone*

Le programme de tri représenté à la figure 4.7 est quadratique, et au moins aussi inefficace que les algorithmes naïfs classiques. Pour obtenir une machine concrète d'efficacité linéaire, il faudra transformer les affectations multiples, non plus en séquences d'affectations simples, mais en ensembles d'affectations simples à exécuter en parallèle, sur des machines distinctes.

Des machines distinctes sont équipées de mémoires disjointes. Par exemple, $(x, y) := (e, f)$ pourra être vu comme l'exécution parallèle de $x := e$ et $y := f$ si e ne dépend que des variables de la machine X exécutant la première affectation, et si f ne dépend que de la machine Y exécutant la seconde affectation.

Dans les mêmes conditions, si on souhaite exécuter l'affectation $x := y$, il faudra réaliser une connexion capable de transférer une information de la machine Y vers la machine X. Sans entrer dans les détails techniques, nous admettons la possibilité de telles connexions. Plus précisément, si C désigne la connexion (ou le *canal*) de Y vers X, nous admettons que l'exécution *simultanée* des instructions spéciales

$$C!y$$

par la machine Y et

$$C?x$$

par la machine X a pour effet global de réaliser l'affectation $x := y$. Aucune signification n'est attachée ici à une instruction spéciale isolée; ces instructions ne peuvent apparaître qu'en paires correspondantes. Ce mécanisme de communication *synchrone* (tout se passe comme si l'émission et la réception d'un "message" étaient simultanées) est effectivement utilisé dans des langages tels que CSP et OCCAM (Hoare 1978 et 1985, May 1983).

4.3.6. *Une réalisation parallèle*

Il est très simple d'obtenir une réalisation du système de tri au moyen du mécanisme de communication synchrone. Le cycle de base du système (Fig. 4.5) est

$$\begin{array}{l} V[0:n-1] := V[1:n]\,;\ V[n] := A[j]\,; \\ \mathit{sort}(1..n)\,; \\ B[j] := V[-n]\,;\ V[-n:-1] := V[1-n:0]\,; \\ \mathit{sort}(1..n)\,; \\ j := j+1\,. \end{array}$$

Il peut se décomposer en le processus parallèle E,

$$\begin{array}{l} D_n!A[j]\,; \\ U_n?B[j]\,; \\ j := j+1\,, \end{array}$$

qui modélise l'*environnement*,[3] et en le processus Q_n,

$$
\begin{array}{l}
V[0:n-1] \;:=\; V[1:n]\;; \\
D_n?V[n]\;; \\
sort(1..n)\;; \\
U_n!V[-n]\;; \\
V[-n:-1] \;:=\; V[1-n:0]\;; \\
sort(1..n)\,,
\end{array}
$$

qui modélise le système de tri proprement dit. Bien plus, le processus Q_n résulte lui-même de la mise en parallèle du processus P_0,

$$D_0?V[0]\;;\; U_0!V[0]\,,$$

et des processus P_i $(i = 1, \ldots, n)$ repris ci-dessous,

$$
\begin{array}{l}
D_{i-1}!V[i]\;; \\
D_i?V[i]\;; \\
sort(i)\;; \\
U_i!V[-i]\;; \\
U_{i-1}?V[-i]\;; \\
sort(i)\,.
\end{array}
$$

Les notations utilisées pour les canaux sont celles de la figure 4.8; chaque processus P_i est associé à la machine élémentaire du même nom.

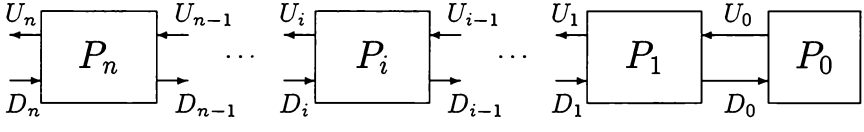

Figure 4.8 *Un tableau de tri systolique*

Remarque. Le système de tri que nous venons de voir fonctionne selon un cycle binaire comportant une phase de communication et une phase de calcul (tri binaire). Il appartient à la classe des tableaux (ou des algorithmes) dits *systoliques* par référence aux deux phases du cycle cardiaque: la systole, ou contraction, et la diastole, ou relaxation.

4.3.7. *Un point de technologie*

Considérons en termes opérationnels un début d'exécution du système parallèle $E \| P_n \| \cdots \| P_0$. Le processus E ne peut envoyer $T[j]$ avant que P_n ne soit prêt à le recevoir; de même, P_{i+1} ne pourra effectuer son premier envoi avant que P_i ne soit prêt à le recevoir. On voit donc que l'opération

[3] c'est-à-dire l'entité émettant le flux d'entrée A et recevant le flux de sortie B.

d'entrée *IN* n'est pas réalisée de manière rigoureusement parallèle; il y a d'abord transmission de P_1 vers P_0, puis de P_2 vers P_1 et ainsi de suite; la transmission du processus E vers le processus P_n clôture l'action *IN*.

Cette situation est-elle acceptable ? Pour que la réponse soit positive, il faut que ces délais restent négligeables à l'échelle globale. Cela sera le cas si n n'est pas trop grand et si une technologie adéquate[4] est utilisée. Si ces conditions ne sont pas remplies, l'efficacité globale de la réalisation parallèle décrite au paragraphe précédent restera quadratique.

On peut résoudre ce problème facilement, en ajoutant à chaque processeur élémentaire P_i $(i = 1, \ldots, n)$ une troisième case mémoire, notée $w[i]$, qui fait office de tampon. Cela permet de synchroniser complètement les transmissions, au prix d'un délai supplémentaire constant. Le processus P_i devient

$w[i] := V[i]$;
$D_{i-1}!w[i] \parallel D_i?V[i]$;
$sort(i)$;
$w[i] := V[-i]$;
$U_i!w[i] \parallel U_{i-1}?V[-i]$;
$sort(i)$.

La solution ainsi obtenue est d'efficacité linéaire.

4.3.8. *Limitations pratiques du parallélisme implicite*

La technique que nous venons d'illustrer est très simple parce qu'elle repose sur une idée élémentaire: l'exécution simultanée de n affectations simples équivaut à l'exécution d'une seule affectation n-uple.

Les limitations de la technique du parallélisme implicite sont celles de l'idée sous-jacente. Nous avons vu en particulier que, bien souvent, l'effet global de l'exécution en parallèle de plusieurs affectations n'était pas déterministe. Dans ce cas, l'assimilation à une seule affectation multiple n'est pas valable. La notion de système formel concurrent, introduite au paragraphe 4.4, apporte une réponse aussi simple que possible à ce problème.

4.3.9. *Puissance théorique du parallélisme implicite*

Le tri systolique et la plupart des applications contemporaines du parallélisme implicite ont pour seul intérêt l'amélioration de l'efficacité des algorithmes classiques. Chaque processus élémentaire accomplit une partie du calcul à réaliser, et communique ses résultats à ses voisins. La communication est ici un auxiliaire du calcul.

[4] pour laquelle le temps de communication d'un processeur vers son voisin est petit en comparaison du temps nécessaire à un calcul élémentaire.

La distinction entre communication et calcul est toutefois moins nette qu'il n'y paraît à première vue. Plus précisément, on peut effectuer des calculs d'une complexité illimitée au moyen d'un réseau de processus extrêmement rudimentaires.

Un *perceptron* est une entité à une sortie et plusieurs entrées. L'unique calcul effectué par le perceptron est la somme pondérée des signaux d'entrée; si cette somme dépasse un certain seuil, le signal de sortie est mis à 1, sinon il est mis à 0. Un réseau de perceptrons adéquatement connectés permet de calculer. Une configuration initiale "code" les données d'un problème et la configuration obtenue après un certain nombre de pas d'exécution est interprétée comme le résultat du calcul. Le réseau est "programmé" par ajustement des coefficients de pondération internes à chaque perceptron. On parle souvent de "réseaux neuronaux", par analogie avec le cerveau humain, qui semble fonctionner d'une manière analogue.

Une variante particulièrement élémentaire de réseau neuronal (ou *automate cellulaire*) est l'*automate de Conway*. Les cellules sont disposées en damier. Chaque cellule n'a que deux états, "actif" et "éteint". L'état d'une cellule A est connu d'elle-même et de ses huit voisins immédiats, qui forment un carré de 3×3 cellules dont A est le centre.

A chaque étape du calcul, chaque cellule consulte son état et celui de ses voisins. Une cellule active reste active si elle est entourée de deux ou de trois cellules actives; elle s'éteint sinon. De même, une cellule éteinte devient active si elle est entourée d'exactement trois cellules actives; elle reste éteinte sinon.

Ce schéma très simple — on peut à peine parler de calcul — permet pourtant, en principe du moins, d'effectuer tout calcul à la portée d'une machine classique.

4.4. Systèmes formels concurrents

4.4.1. *Introduction*

Dans la plupart des cas, le parallélisme ne peut pas être traité de manière implicite, et l'approche adoptée au paragraphe précédent pour le problème du tri systolique ne sera plus possible. Comme nous ne voulons pas, pour les raisons invoquées au paragraphe 4.1, introduire un opérateur de composition parallèle, nous quittons l'approche structurée pour revenir à l'approche plus primitive des systèmes formels séquentiels introduite au paragraphe 3.4. Le formalisme SFS s'adapte immédiatement au parallélisme comme suit : on remplace le processus P par un ensemble $\mathcal{P}$ de processus. Pour éviter toute confusion, les étiquettes appartenant à des processus distincts sont supposées distinctes; autrement dit, les processus sont des ensembles disjoints. Cette extension donne lieu au formalisme SFC (système formel concurrent). Les différences par rapport à SFS sont minimes et ne justifient pas une présentation détaillée de SFC. Le seul point à souligner est que l'origine et l'extrémité

d'une transition (§ 3.4.1) spécifient non seulement un changement d'état de contrôle, mais aussi l'identité du ou des processus responsables de l'exécution de la transition. Nous verrons au paragraphe 4.5 l'utilité des transitions exécutées conjointement par plusieurs processus, mais nous considérons d'abord un exemple plus simple.

4.4.2. *Problème de l'exclusion mutuelle*

Un problème classique en programmation parallèle est celui de *l'exclusion mutuelle.* Il survient lorsque plusieurs processus doivent, de temps à autre, effectuer une même opération qui, par nature, ne peut être effectuée que par un processus à la fois. Considérons un cas concret et fréquent. Un gros ordinateur fonctionne usuellement en temps partagé: plusieurs processus, appartenant à différents utilisateurs, sont exécutés simultanément. Ceci a deux avantages. D'abord, les ressources de la machine sont mieux exploitées. Pendant que l'un des processus utilise l'unité arithmétique et logique, un autre peut commander un transfert d'information entre la mémoire et un périphérique, par exemple. Ensuite, pour peu que la charge globale ne soit pas excessive, chaque utilisateur aura l'impression de disposer de l'ordinateur pour lui seul. Toutefois, le système d'exploitation de la machine doit garantir que les commandes émanant des différents processus n'interfèrent pas. Si par exemple deux processus demandent l'accès à un même périphérique, le système d'exploitation devra veiller à ce que ces accès se fassent strictement en séquence, ce qui pourra impliquer la mise en attente temporaire de l'un des processus demandeurs.

Le problème de l'exclusion mutuelle survient dans les contextes les plus variés, et beaucoup d'autres problèmes importants en programmation parallèle en dérivent assez directement. Pour cette raison, nous donnerons dans ce chapitre trois solutions particulières à ce problème.

4.4.3. *Exclusion mutuelle entre deux processus*

La figure 4.9 représente deux processus *cycliques*, c'est-à-dire deux processus dont l'exécution n'est pas censée se terminer. Chacun de ces processus comporte deux points de contrôle, correspondant à deux états possibles. En section non critique, les processus n'interfèrent pas, tandis qu'en section critique, ils accèdent à une ressource partagée unique. Cela signifie que les deux processus ne peuvent pas se trouver simultanément en section critique.

Cette propriété d'exclusion mutuelle est une propriété d'invariance; elle est formalisée par l'assertion $\neg at\,(p_c, q_c)$.

Remarque. La formule $at\,(p_c, q_c)$ peut être vue comme une abréviation de $(at\,p_c \wedge at\,q_c)$.

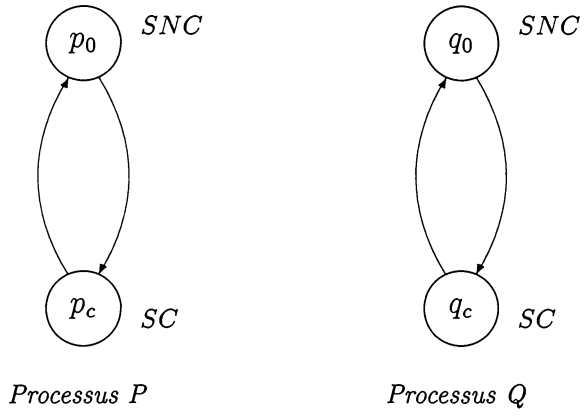

Figure 4.9 *Deux processus cycliques*

4.4.4. *Un algorithme naïf*

Le système formel concurrent représenté graphiquement à la figure 4.10 est une première solution au problème de l'exclusion mutuelle.

La variable T (pour "Tour") est partagée entre les deux processus; elle détermine quel processus a le droit d'accéder à sa section critique. Un invariant du système est la conjonction des quatre assertions données ci-dessous :

$$\begin{aligned} &at\ p_w \ \Rightarrow\ (T = q \vee at\ q_w)\,,\\ &at\ q_w \ \Rightarrow\ (T = p \vee at\ p_w)\,,\\ &at\ p_c \ \Rightarrow\ (T = p \wedge \neg at\ q_0)\,,\\ &at\ q_c \ \Rightarrow\ (T = q \wedge \neg at\ p_0)\,. \end{aligned}$$

Les conséquences logiques de cet invariant sont les propriétés d'invariance de l'algorithme. L'une d'elles est l'assertion

$$(at\ p_c \ \Rightarrow\ T = p) \wedge (at\ q_c \ \Rightarrow\ T = q)\,,$$

qui exprime bien le rôle de la variable T et l'exclusion mutuelle. Pour peu que l'exécution du système parte d'un état compatible avec l'invariant (par exemple, les deux processus sont dans leur section non critique), on est assuré qu'à aucun moment ultérieur les deux processus ne seront simultanément en section critique. Cet algorithme est très simple mais a le défaut rédhibitoire d'imposer une stricte alternance : chaque processus accède à son tour à sa section critique.

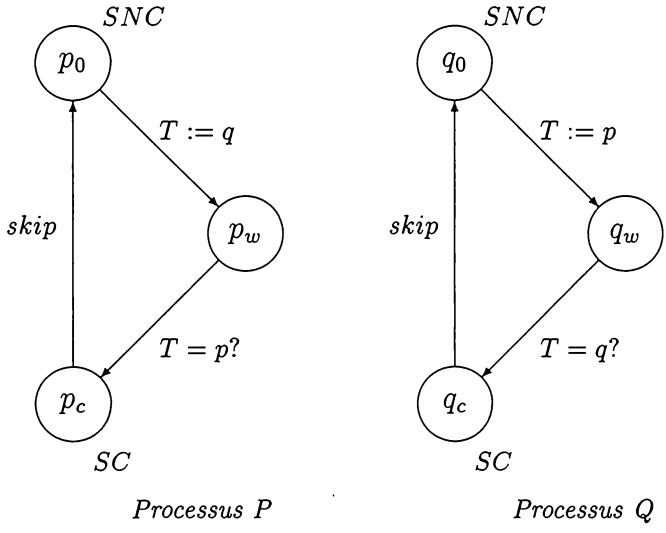

Figure 4.10 *Un algorithme naïf*

4.4.5. *Une solution moins rigide*

Le système formel concurrent représenté graphiquement à la figure 4.10 n'imposera plus la stricte alternance si on remplace les tests $T = p$? et $T = q$? respectivement par $(T = p \vee at\ q_0)$? et $(T = q \vee at\ p_0)$?. Si l'un des processus est en section non critique et y reste, l'autre processus a le droit d'accéder sans restriction à sa section critique.

Un invariant approprié de ce système est

$$\begin{array}{ll} & [at\ p_c \ \Rightarrow\ (T = p \vee at\ q_0)] \\ \wedge & [at\ q_c \ \Rightarrow\ (T = q \vee at\ p_0)]\,. \end{array}$$

Cette solution n'est pas réalisable telle quelle, parce qu'elle implique un test sur un prédicat de contrôle. Le formalisme des systèmes formels (séquentiels ou parallèles) n'admet que le test d'expressions booléennes construites avec des *variables* du programme. Nous allons donc introduire de nouvelles variables qui indiqueront la valeur de vérité des prédicats de contrôle $at\ p_0$ et $at\ q_0$; la définition de ces variables sera incluse dans l'invariant.

Dans le formalisme SFC, cet algorithme sera représenté par le triplet $(\mathcal{P}, \mathcal{M}, \mathcal{T})$ où

$$\mathcal{P} ::= \{P, Q\}\,;\ P = \{p_0, p_w, p_c\}\,,\ Q = \{q_0, q_w, q_c\}\,;$$

$$\mathcal{M} ::= \{inP, inQ, T\}\,;$$

$$\begin{aligned}\mathcal{T} ::= \{\ & (p_0,\ (T, inP) := (q, vrai)\ ,\ p_w)\ , \\ & (p_w,\ T = p \vee \neg inQ \longrightarrow skip\ ,\ p_c)\ , \\ & (p_c,\ inP := faux\ ,\ p_0)\ , \\ & (q_0,\ (T, inQ) := (p, vrai)\ ,\ q_w)\ , \\ & (q_w,\ T = q \vee \neg inP \longrightarrow skip\ ,\ q_c)\ , \\ & (q_c,\ inQ := faux\ ,\ q_0)\ \}\ .\end{aligned}$$

On obtient ainsi le système représenté à la figure 4.11. Comme d'habitude, on établira la propriété d'exclusion mutuelle par un invariant, qui sera la formule

$$\begin{aligned}& (at\ p_0 \equiv \neg inP) \wedge (at\ q_0 \equiv \neg inQ) \wedge \\ & (at\ p_c \Rightarrow (T = p \vee \neg inQ)) \wedge \\ & (at\ q_c \Rightarrow (T = q \vee \neg inP))\ ,\end{aligned}$$

dont la conséquence logique est l'assertion

$$at(p_c, q_c) \Rightarrow (T = p \wedge T = q)\ ,$$

ou encore

$$\neg at(p_c, q_c)\ .$$

4.4.6. *Problème du grain*

Le dernier algorithme que nous avons obtenu est satisfaisant en principe mais délicat à réaliser. Les transitions $(p_0 \rightarrow p_w)$ et $(q_0 \rightarrow q_w)$ comportent

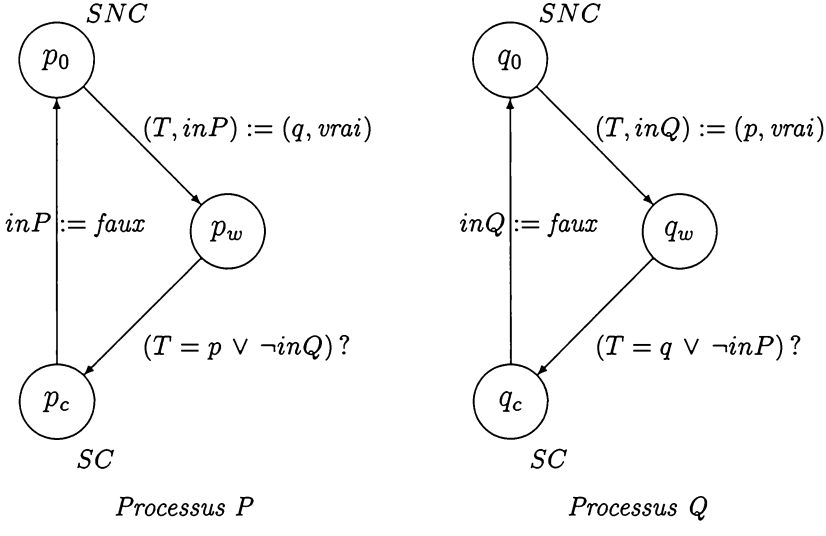

Figure 4.11 *Introduction de variables*

des affectations doubles alors que, en pratique, seules des affectations simples peuvent être considérées comme atomiques. Le remède à appliquer ici est clair : il faut scinder les affectations doubles en séquences de deux affectations simples.

Plus précisément, dans le processus P, l'affectation

$$(T, inP) := (q, vrai)$$

sera décomposée en

$$T := q\,;\ inP := vrai$$

ou en

$$inP := vrai\,;\ T := q\,.$$

Rien ne prouve que l'une et l'autre de ces décompositions soient correctes. On pourrait penser que, pour préserver la propriété d'invariance

$$at\ q_c \ \Rightarrow\ (T = q \ \vee\ \neg inP)\,,$$

il serait opportun d'affecter T avant inP. Nous envisageons donc d'abord la première décomposition, que nous appliquons uniquement au processus P. Le résultat de cette décomposition est représenté graphiquement à la figure 4.12.

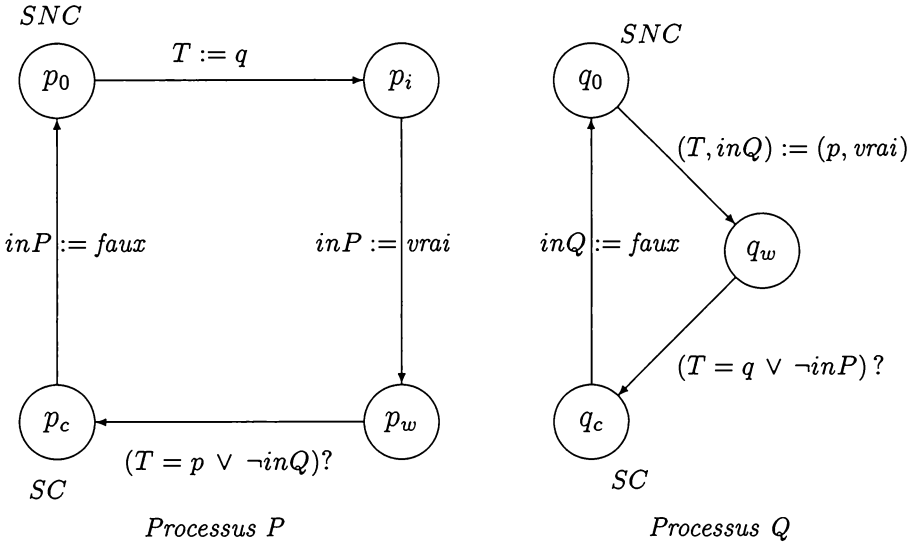

Figure 4.12 *Décomposition incorrecte*

Observons l'apparition d'une nouvelle étiquette p_i dans le processus P : un point de contrôle intermédiaire est induit par la décomposition.

Cette "solution" n'est pas correcte, car elle ne respecte pas l'exclusion mutuelle. Pour le voir, il suffit de construire une exécution fautive, comme celle de la figure 4.13.

p_0	q_0	$\neg inP \wedge \neg inQ$
p_i	q_0	$\neg inP \wedge \neg inQ \wedge T = q$
p_i	q_w	$\neg inP \wedge inQ \wedge T = p$
p_i	q_c	$\neg inP \wedge inQ \wedge T = p$
p_w	q_c	$inP \wedge inQ \wedge T = p$
p_c	q_c	!!!

Figure 4.13 *Exécution fautive*

4.4.7. *L'algorithme de Peterson*

Si la première décomposition est incorrecte, il est possible (mais non certain) que la seconde décomposition soit effectivement correcte. C'est bien le cas en l'occurrence, et la double décomposition donne lieu à l'algorithme représenté à la figure 4.14. Cet algorithme, quoique très simple, n'a été découvert que tardivement (Peterson 1981).

Pour prouver l'exactitude de cet algorithme, nous devons comme d'habitude construire un invariant qui implique l'exclusion mutuelle. Un invariant adéquat est donné à la figure 4.16. Une preuve plus subtile, évitant la construction de l'invariant, se trouve dans (Feijen, 1989). L'algorithme de Peterson est représenté sous la forme d'un système formel concurrent à la figure 4.15.

Pour vérifier que l'invariant I implique bien l'exclusion mutuelle, il suffit de vérifier que $I[at\ p_c q_c]$ est identiquement faux. On calcule d'abord

$$\begin{aligned} I[at\ p_c] = \ & (faux \ \Rightarrow \ (T = q \ \vee \ at\ q_w)) \\ & \wedge \ (at\ q_w \ \Rightarrow \ (T = p \ \vee \ faux)) \\ & \wedge \ (vrai \ \Rightarrow \ (T = p \ \vee \ \neg inQ \ \vee \ at\ q_i)) \\ & \wedge \ (at\ q_c \ \Rightarrow \ (T = q \ \vee \ \neg inP \ \vee \ faux)) \\ & \wedge \ (faux \ \equiv \ \neg inP) \\ & \wedge \ (at\ q_0 \ \equiv \ \neg inQ)\,. \end{aligned}$$

Cette assertion se réduit en

$$\begin{aligned} I[at\ p_c] = \ & (at\ q_w \ \Rightarrow \ T = p) \ \wedge \ (T = p \ \vee \ \neg inQ \ \vee \ at\ q_i) \\ & \wedge \ (at\ q_c \ \Rightarrow \ T = q) \ \wedge \ inP \ \wedge \ (at\ q_0 \ \equiv \ \neg inQ)\,. \end{aligned}$$

On calcule ensuite

$$\begin{aligned} I[at\ p_c q_c] = \ & (faux \ \Rightarrow \ T = p) \ \wedge \ (T = p \ \vee \ \neg inQ \ \vee \ faux) \\ & \wedge \ (vrai \ \Rightarrow \ T = q) \ \wedge \ inP \ \wedge \ (faux \ \equiv \ \neg inQ)\,. \end{aligned}$$

Cette assertion se réduit à la formule

$$I[at\ p_c q_c] = \ (T = p \ \wedge \ T = q \ \wedge \ inP \ \wedge \ inQ)\,,$$

qui est identiquement fausse.

Remarque. On écrit indifféremment $(at\ p_c \wedge at\ q_c)$ ou $at\ (p_c, q_c)$ ou $at\ p_c q_c$. De même, $I[at\ p_c q_c]$ est une abréviation de $I[at\ p_c][at\ q_c]$.

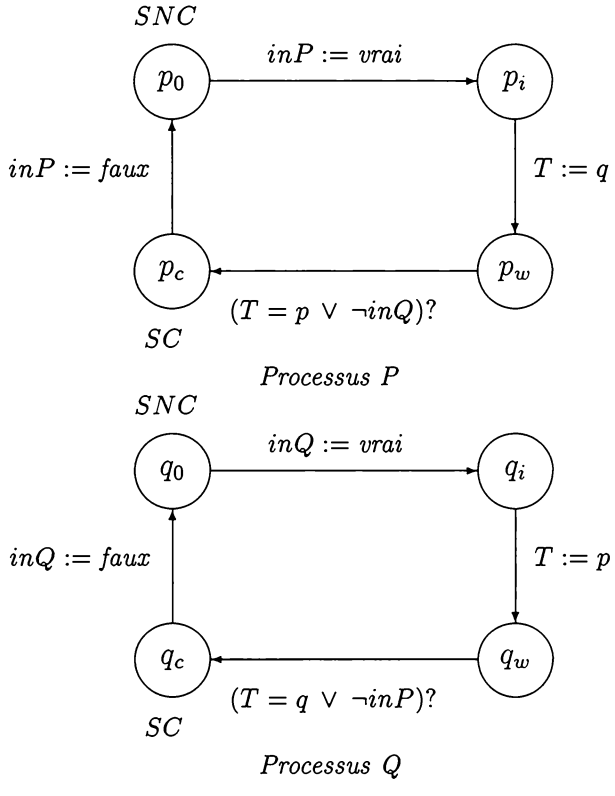

Figure 4.14 *Algorithme de Peterson*

4.5. Exclusion mutuelle entre n processus

Nous considérons maintenant le problème de l'exclusion mutuelle au sein d'un ensemble de n processus cycliques.

Un processus pourra, a priori, se trouver dans trois états de contrôle : l'état non critique, l'état d'attente et l'état critique. Dans l'état non critique, il pourra réaliser diverses tâches n'impliquant pas l'exclusion mutuelle. Lorsque l'exclusion mutuelle devient nécessaire, le processus le signale et passe dans l'état d'attente, jusqu'à ce que son "tour" arrive. Le processus passe alors dans l'état critique et exécute la tâche nécessitant l'exclusion mutuelle (appelée *section critique*). A la fin de cette exécution, le processus signale qu'il a fini, et retourne dans l'état non critique.

On peut modéliser cette situation au moyen d'un système formel concurrent, composé de n processus (désignés par les lettres minuscules $p, q, r, \ldots$). Un

$$
\begin{aligned}
\mathcal{P} ::= \ & \{P, Q\}\,,\ P = \{p_0, p_i, p_w, p_c\}\,,\ Q = \{q_0, q_i, q_w, q_c\} \\
\mathcal{M} ::= \ & \{inP, inQ, T\} \\
\mathcal{T} ::= \ & \{\,(p_0,\ inP := vrai\,,\ p_i) \\
& \quad (p_i,\ T := q\,,\ p_w) \\
& \quad (p_w,\ T = p \vee \neg inQ \longrightarrow skip\,,\ p_c) \\
& \quad (p_c,\ inP := faux\,,\ p_0) \\
& \quad (q_0,\ inQ := vrai\,,\ q_i) \\
& \quad (q_i,\ T := p\,,\ q_w) \\
& \quad (q_w,\ T = q \vee \neg inP \longrightarrow skip\,,\ q_c) \\
& \quad (q_c,\ inQ := faux\,,\ q_0)\ \}
\end{aligned}
$$

Figure 4.15 *Algorithme de Peterson*

$$
\begin{aligned}
I = \ & (at\ p_w \ \Rightarrow\ (T = q \vee at\ q_w)) \\
\wedge\ & (at\ q_w \ \Rightarrow\ (T = p \vee at\ p_w)) \\
\wedge\ & (at\ p_c \ \Rightarrow\ (T = p \vee \neg inQ \vee at\ q_i)) \\
\wedge\ & (at\ q_c \ \Rightarrow\ (T = q \vee \neg inP \vee at\ p_i)) \\
\wedge\ & (at\ p_0 \ \equiv\ \neg inP) \\
\wedge\ & (at\ q_0 \ \equiv\ \neg inQ)\,.
\end{aligned}
$$

Figure 4.16 *Invariant de l'algorithme de Peterson*

processus p comportera l'état non critique p_0, l'état d'attente p_w et l'état critique p_c. Le processus se trouve initialement dans l'état non critique. Tant que l'accès à la section critique n'est pas nécessaire, il effectue une certaine tâche dont le détail ne nous importe pas. Ceci est modélisé par une transition du type $(p_0 \rightarrow p_0)$. Quand l'accès devient nécessaire, deux possibilités existent. Si aucun autre processus ne se trouve en section critique, le processus p peut y entrer sans délai, ce qui correspond à l'exécution d'une transition du type $(p_0 \rightarrow p_c)$. Si au contraire un autre processus se trouve en section critique, le processus p devra se mettre en attente, ce qui correspond à l'exécution d'une transition du type $(p_0 \rightarrow p_w)$. Supposons que le processus p soit dans l'état critique. Il accomplit sa tâche, ce qui revient à exécuter un certain nombre de fois une transition du type $(p_c \rightarrow p_c)$. Quand cette tâche est terminée, si aucun processus n'est en attente, le processus p retourne dans l'état non critique et une transition de type $(p_c \rightarrow p_0)$ est exécutée. Si au contraire, un autre processus, par exemple le processus q, est en attente, une transition du type $(p_c q_w \rightarrow p_0 q_c)$ sera exécutée.

Remarque. Du point de vue formel, un processus n'est rien d'autre qu'un ensemble non vide d'étiquettes. En réalité, un processus est un programme, donc un outil de commande pour un ordinateur. Une transition décrit une commande élémentaire, et indique aussi le ou les processus responsable(s)

de son exécution. Par exemple, la transition $(p_0 \rightarrow p_0)$ ne concerne que le processus p, tandis que la transition $(p_c q_w \rightarrow p_0 q_c)$ est exécutée conjointement par les deux processus p et q.

4.5.1. *Une solution élémentaire*

Dans la vie courante, les situations requérant l'exclusion mutuelle surviennent souvent. Par exemple, à tout instant, un seul client est admis à une caisse de magasin; de même, les gens montent dans un autobus un par un. Dans ces deux cas, la situation est modélisée de façon fort simple. Une personne est en "section critique" et d'autres forment une file d'attente (ou, parfois, un ensemble désordonné).

Ceci se transpose aisément à notre problème. On se donne une variable SC (pour "Section Critique") et une variable E (pour "Ensemble d'attente"). A tout instant, la valeur de SC est le nom du processus en section critique; quand aucun processus ne s'y trouve, cette valeur est conventionnellement 0. L'ensemble E contient les noms des processus en attente, s'il y en a.

Les différents types de transitions du système, déjà évoqués au paragraphe précédent, peuvent maintenant être explicités. Si p et q sont deux processus distincts, alors le système comportera les transitions

1. $(p_0,\ \neg rcs_p \longrightarrow [\,]\,,\ p_0)$,
2. $(p_0,\ rcs_p \wedge SC = 0 \longrightarrow SC := p\,,\ p_c)$,
3. $(p_0,\ rcs_p \wedge SC \neq 0 \longrightarrow E := E \cup \{p\}\,,\ p_w)$,
4. $(p_c,\ rcs_p \longrightarrow [\,]\,,\ p_c)$,
5. $(p_c q_w,\ \neg rcs_p \wedge q \in E \longrightarrow (SC, E) := (q, E \setminus \{q\})\,,\ p_0 q_c)$,
6. $(p_c,\ \neg rcs_p \wedge E = \emptyset \longrightarrow SC := 0\,,\ p_0)$.

La notation $[\,]$ représente une action interne de l'un des processus. Il est inutile de modéliser plus avant cette action, puisqu'elle n'influe que sur la mémoire propre du processus qui l'exécute, mémoire propre qui n'est elle-même pas modélisée. Les actions internes relatives au processus p peuvent cependant modifier la valeur de la condition rcs_p. Cette condition est vraie aux instants où l'accès du processus p à sa section critique est nécessaire.

Les transitions 1 et 4 représentent l'activité interne du processus p dans les états non critique et critique respectivement. Les deuxième et troisième transitions correspondent à la demande d'accès : le processus p demande l'accès et l'obtient immédiatemment (transition 2) ou se met en attente (transition 3). Enfin, les deux dernières transitions concernent le retour dans l'état non critique. Lors de ce retour, un processus en attente passe en section critique (transition 5); si aucun processus n'est en attente, la transition 6 est exécutée.

Ce programme est intuitivement correct; pour obtenir une preuve rigoureuse, il suffit de formaliser son fonctionnement en un invariant :

$$\begin{aligned} I \;=_{def}\quad & \forall r\,[\,at\,r_w \;\equiv\; r \in E\,] \\ \wedge\quad & \forall r\,[\,at\,r_c \;\equiv\; SC = r\,] \\ \wedge\quad & (SC = 0 \;\Rightarrow\; E = \emptyset)\,. \end{aligned}$$

La première ligne exprime que les processus en attente sont répertoriés dans l'ensemble d'attente; la deuxième ligne exprime que le processus en section critique est celui indiqué par la variable SC et la troisième ligne exprime que la ressource partagée est bien utilisée: elle ne peut être inutilisée que quand aucun processus n'est en attente.

4.5.2. *Introduction d'un contrôleur*

L'algorithme que nous venons de présenter est très simple, mais sa mise en œuvre n'est pas évidente. En effet, les variables SC et E sont communes à tous les processus, et ces processus peuvent être exécutés sur des machines distinctes, voire fort distantes l'une de l'autre. Il serait préférable d'éviter ce partage de variables et de créer un processus spécial, le *contrôleur*, qui aurait seul la charge de gérer ces deux variables. Nous enrichissons donc notre système d'un processus spécial nommé *ctrl*, dont au départ l'unique étiquette est C. Les transitions du système sont récrites pour mettre en évidence le rôle du contrôleur; si p et q désignent des processus distincts, l'ensemble $\mathcal{T}$ des transitions du système comportera les éléments suivants:

$$\begin{aligned} &(p_0,\ \neg rcs_p \longrightarrow [\,]\,,\ p_0)\,, \\ &(p_0C,\ rcs_p \wedge SC = 0 \longrightarrow SC := p\,,\ p_cC)\,, \\ &(p_0C,\ rcs_p \wedge SC \neq 0 \longrightarrow E := E \cup \{p\}\,,\ p_wC)\,, \\ &(p_c,\ rcs_p \longrightarrow [\,]\,,\ p_c)\,, \\ &(p_cq_wC,\ \neg rcs_p \wedge q \in E \longrightarrow (SC, E) := (q, E \setminus \{q\})\,,\ p_0q_cC)\,, \\ &(p_cC,\ \neg rcs_p \wedge E = \emptyset \longrightarrow SC := 0\,,\ p_0C)\,. \end{aligned}$$

Bien évidemment, cette transformation mineure ne fait qu'indiquer l'existence et le rôle du nouveau processus $ctrl = \{C\}$. La dynamique du système n'est en rien modifiée, et l'invariant subsiste sans modification.

4.5.3. *Une version plus réaliste*

Considérons d'un peu plus près la transition

$$(p_0C,\ rcs_p \wedge SC = 0 \longrightarrow SC := p\,,\ p_cC)\,.$$

On peut opérationnellement l'interpréter comme suit. Le processus p doit accéder à sa section critique (rcs_p est vrai) et, comme la ressource partagée est libre (SC vaut 0), l'accès est immédiatement autorisé par le contrôleur. On discerne bien que la transition modélise en fait deux actions: une requête du processus p et une autorisation d'accès donnée par le contrôleur. Notre système sera plus réaliste s'il modélise deux actions bien distinctes par deux transitions plutôt qu'une seule; le cas de la transition $(p_0C \rightarrow p_wC)$ est analogue. On remplace donc les transitions

$$(p_0C,\ rcs_p \wedge SC = 0 \longrightarrow SC := p\ ,\ p_cC)\,,$$
$$(p_0C,\ rcs_p \wedge SC \neq 0 \longrightarrow E := E \cup \{p\}\ ,\ p_wC)$$

par les transitions

$$(p_0C,\ rcs_p \longrightarrow skip\ ,\ p_wC_p)\,,$$
$$(p_wC_p,\ SC = 0 \longrightarrow SC := p\ ,\ p_cC)\,,$$
$$(p_wC_p,\ SC \neq 0 \longrightarrow E := E \cup \{p\}\ ,\ p_wC)\,.$$

La première transition modélise l'envoi de la requête, la deuxième modélise l'autorisation immédiate d'accès et la troisième transition modélise la mise en attente du processus demandeur. Le contrôleur comporte maintenant les étiquettes $C, C_p, C_q, \ldots$ Le prédicat *at* C est vrai quand le contrôleur est prêt à recevoir une requête; *at* C_p est vrai quand le contrôleur a reçu une requête émanant du processus p et qu'il doit y répondre.

Cette transformation du programme altère l'invariant; la première ligne

$$at\ p_w\ =\ p \in E$$

est à remplacer par

$$at\ p_w\ =\ (at\ C_p + p \in E)\,.$$

En effet, quand *at* C_p est vrai, le processus p est en attente mais p n'est pas (ou pas encore) un élément de l'ensemble d'attente.

Remarque. Rappelons qu'il est fréquent d'assimiler *faux* à 0 et *vrai* à 1. Avec cette convention, si $A_1, \ldots, A_n$ sont des formules et si k est un entier naturel, la formule

(4.1) $\qquad A_1 + \cdots + A_n = k$

signifie qu'exactement k formules de la liste $(A_1, \ldots, A_n)$ sont vraies.[5]
(La formule 4.1 ne peut être vraie que si k est un élément de $\{0, 1, \ldots, n\}$. On peut aussi écrire $A_1 + \cdots + A_n = B_1 + \cdots + B_p$ pour exprimer qu'il y a le même nombre de formules vraies dans les listes $(A_1, \ldots, A_n)$ et $(B_1, \ldots, B_p)$.)

Le même souci de réalisme conduit à remplacer les transitions

$$(p_cq_wC,\ \neg rcs_p \wedge q \in E \longrightarrow (SC, E) := (q, E \setminus \{q\})\ ,\ p_0q_cC)\,,$$
$$(p_cC,\ \neg rcs_p \wedge E = \emptyset \longrightarrow SC := 0\ ,\ p_0C)$$

par les transitions

$$(p_cq_wC,\ \neg rcs_p \longrightarrow skip\ ,\ p_0q_wC'_p)\,,$$
$$(p_0q_wC'_p,\ q \in E \longrightarrow (SC, E) := (q, E \setminus \{q\})\ ,\ p_0q_cC)\,,$$
$$(p_cC,\ \neg rcs_p \longrightarrow skip\ ,\ p_0C'_p)\,,$$
$$(p_0C'_p,\ E = \emptyset \longrightarrow SC := 0\ ,\ p_0C)\,.$$

[5] Il peut paraître inapproprié d'assimiler des notions aussi différentes que des valeurs de vérité et des nombres entiers. Cela se justifie ici parce que cette assimilation permet de donner à certaines formules une forme nettement plus concise. Rappelons que la longueur est l'inconvénient principal des développements formels.

Intuitivement, $at\ C'_p$ est vrai quand le contrôleur a reçu du processus p un avis de fin d'accès à la ressource partagée, mais n'a pas encore réagi à cet avis.

L'invariant doit à nouveau être modifié; il devient

$$\begin{aligned} & \forall r\,[\,at\ r_w\ =\ (at\ C_r + r \in E)\,] \\ \wedge\ & \forall r\,[\,(at\ r_c + at\ C'_r)\ =\ (SC = r)\,] \\ \wedge\ & (SC = 0\ \Rightarrow\ E = \emptyset)\,. \end{aligned} \tag{4.2}$$

Remarquons enfin que les transitions

$$\begin{aligned} & (p_c q_w C,\ \neg rcs_p\ \longrightarrow\ skip\,,\ p_0 q_w C'_p)\,, \\ & (p_0 q_w C'_p,\ q \in E\ \longrightarrow\ (SC, E) := (q, E \setminus \{q\})\,,\ p_0 q_c C)\,, \\ & (p_0 C'_p,\ E = \emptyset\ \longrightarrow\ SC := 0\,,\ p_0 C)\,, \end{aligned}$$

n'impliquent pas réellement tous les processus qu'elles évoquent. Plus formellement elles peuvent être, sans nouvelle altération de l'invariant, remplacées par les transitions

$$\begin{aligned} & (p_c C,\ \neg rcs_p\ \longrightarrow\ skip\,,\ p_0 C'_p)\,, \\ & (q_w C'_p,\ q \in E\ \longrightarrow\ (SC, E) := (q, E \setminus \{q\})\,,\ q_c C)\,, \\ & (C'_p,\ E = \emptyset\ \longrightarrow\ SC := 0\,,\ C)\,. \end{aligned}$$

4.5.4. *Propriétés du programme*

Rappelons qu'une formule I est un invariant du système $\mathcal{S} = (\mathcal{P}, \mathcal{M}, \mathcal{T})$ si et seulement si pour toute transition $\tau \in \mathcal{T}$, le triplet de Hoare $\{I\}\,\tau\,\{I\}$ est vrai. (On dit que τ *respecte* I.) A titre d'exemple, nous allons vérifier que la transition

$$\tau\ =\ (q_w C'_p,\ q \in E\ \longrightarrow\ (SC, E) := (q, E \setminus \{q\})\,,\ q_c C)$$

respecte l'invariant 4.2. D'après la règle de réduction des triplets de Hoare, le triplet $\{I\}\,\tau\,\{I\}$ est équivalent au triplet

$$\{I[at\ q_w C'_p]\}\ q \in E\ \longrightarrow\ (SC, E) := (q, E \setminus \{q\})\ \{I[at\ q_c C]\}\,.$$

On calcule d'abord la précondition $I[at\ q_w C'_p]$; le résultat est

$$\begin{aligned} & SC = p \wedge q \in E \\ \wedge\ & \forall r \neq q\,[\,at\ r_w\ =\ r \in E\,] \\ \wedge\ & \forall r \neq q\,[\,\neg at\ r_c\,]\,. \end{aligned}$$

La postcondition $I[at\ q_c C]$ est

$$\begin{aligned} & SC = q \wedge q \notin E \\ \wedge\ & \forall r \neq q\,[\,at\ r_w\ =\ r \in E\,] \\ \wedge\ & \forall r \neq q\,[\,\neg at\ r_c\,]\,. \end{aligned}$$

La vérification du triplet est alors immédiate. La vérification pour les autres transitions est analogue.

L'invariant étant établi, il peut être utilisé pour démontrer les propriétés d'invariance utiles du programme. Il y a bien sûr l'exclusion mutuelle. Une conséquence logique de l'invariant est l'assertion

$$\forall r\,(at\ r_c \ \Rightarrow\ SC = r)\,.$$

On en déduit immédiatement qu'à tout instant au plus un processus peut être dans sa section critique, puisqu'une variable n'a, à un instant donné, qu'une seule valeur.

Une autre propriété utile est *l'absence d'interblocage.* Un interblocage survient lorsqu'il n'est plus possible d'exécuter aucune transition. On peut vérifier que, dans le cas présent, cela ne se produit jamais. Plus généralement, il est intéressant d'examiner les causes possibles de blocage d'un processus. Le contrôleur ne peut se bloquer que quand *at C* est vrai. Dans cet état, il restera inactif jusqu'à ce qu'un des processus envoie une demande d'accès ou un avis de fin. Par contre, il n'est pas certain qu'un processus p en attente d'accès à sa section critique finira par l'obtenir. Il se pourrait d'abord qu'un autre processus q "monopolise" l'accès à la ressource partagée et reste indéfiniment dans sa section critique. En effet, aucun mécanisme n'est prévu pour limiter le temps pendant lequel *at* q_c est vrai. Une autre possibilité serait que la valeur p, élément de E, soit systématiquement ignorée lorsqu'une valeur est extraite de l'ensemble E. Pour que le système fonctionne, il faut que E soit géré de manière équitable. Une manière simple et naturelle consiste à ordonner l'ensemble E en une file d'attente; cette structure fonctionne selon le principe "premier entré, premier sorti".

4.5.5. *Une version synchrone*

Une transition telle que

$$(p_c C,\ \neg rcs_p \ \longrightarrow\ skip\ ,\ p_0 C'_p)$$

implique une coopération entre les processus p et *ctrl.* Plus précisément, le processus p communique au contrôleur que son accès à la ressource partagée est achevé. Nous avons vu au paragraphe 4.3.5 que ce type de communication pouvait être spécifié plus explicitement au moyen de canaux. Pour chaque station p, on peut imaginer un canal REQ_p, orienté de p vers le contrôleur et destiné à la transmission des requêtes. De même, un canal OK_p transmettra les autorisations du contrôleur vers le processus p, et un canal END_p transmettra en sens inverse les avis de fin.

Il est maintenant possible, grâce aux instructions spéciales de communication liées à ces canaux, d'isoler les rôles du processus p et du contrôleur. Les "pseudo-transitions"

$$(p_0,\ \neg rcs_p \longrightarrow [\,]\,,\ p_0)\,,$$
$$(p_0,\ rcs_p \longrightarrow REQ_p!\,,\ p_w)\,,$$
$$(p_w,\ OK_p?\,,\ p_c)\,,$$
$$(p_c,\ rcs_p \longrightarrow [\,]\,,\ p_c)\,,$$
$$(p_c,\ \neg rcs_p \longrightarrow END_p!\,,\ p_0)$$

sont du ressort exclusif du processus p, tandis que les pseudo-transitions

$$(C,\ REQ_q?\,,\ C_q)\,,$$
$$(C_q,\ SC = 0 \longrightarrow SC := q\,,\ C)\,,$$
$$(C_q,\ SC \neq 0 \longrightarrow E := E \cup \{q\}\,,\ C)\,,$$
$$(C,\ END_q?\,,\ C'_q)\,,$$
$$(C'_q,\ E = \emptyset \longrightarrow SC := 0\,,\ C)\,,$$
$$(C'_q,\ E = \{r\} \cup E' \longrightarrow (SC, E := r, E');\ OK_r!\,,\ C)$$

sont du ressort exclusif du contrôleur. Rappelons cependant qu'il ne s'agit là que d'un simple changement de notation. En effet, une instruction du type $C!$ n'a de signification que conjointement avec une instruction du type $C?$. Par exemple, les deux pseudo-transitions

$$(p_0,\ rcs_p \longrightarrow REQ_p!\,,\ p_w)\,,$$
$$(C,\ REQ_p?\,,\ C_p)$$

n'ont de signification qu'en association l'une avec l'autre; elles équivalent à la transition

$$(p_0C,\ rcs_p \longrightarrow skip\,,\ p_wC_p)\,.$$

Remarque. Les instructions spéciales $C!e$ et $C?x$ ont un double rôle de synchronisation et de transfert d'information; elles réalisent conjointement l'affectation $x := e$. Au contraire, les instructions spéciales $C!$ et $C?$ n'ont qu'un rôle de synchronisation; elles réalisent l'opération *skip*.

Nous récrivons maintenant les programmes correspondant au processus p et au contrôleur dans des notations plus classiques (Hoare, 1978); si P est un programme, la notation $*[P]$ désigne ici le programme consistant à répéter P autant que possible:

$$\begin{array}{l} *[\,*[\neg rcs_p \longrightarrow [\,]\,]\,; \\ \quad REQ_p!\,; \\ \quad OK_p?\,; \\ \quad *[rcs_p \longrightarrow [\,]\,]\,; \\ \quad END_p! \\ ]\,, \end{array}$$

$$
\begin{array}{l}
*[\,\Box_p\ REQ_p? \longrightarrow\ [\,SC = 0 \longrightarrow SC := p;\ OK_p! \\
\qquad\qquad\qquad\qquad\ \Box SC \neq 0 \longrightarrow E := E \cup \{p\}] \\
\quad \Box_p\ END_p? \longrightarrow\ [\,E = \emptyset \longrightarrow SC := 0 \\
\qquad\qquad\qquad\qquad\ \Box\Box_q\ q \in E \longrightarrow OK_q!;\ (SC, E) := (q,\ E \backslash \{q\})] \\
\,]\,.
\end{array}
$$

4.5.6. *Une version asynchrone*

Comme son nom l'indique, le mode de communication synchrone exige la collaboration *simultanée* des correspondants. C'est à la fois un avantage et un inconvénient; par exemple, le téléphone permet de transmettre rapidement de l'information ... à condition que les deux correspondants soient disponibles en même temps. De plus, le téléphone coûte relativement cher. Ceci explique qu'un mode *asynchrone* de communication, tel le courrier postal, soit parfois préféré. Il est moins cher pour de grandes quantités d'information, et ne requiert pas l'attention simultanée des correspondants. Ce qui est vrai pour la communication humaine l'est également pour la communication entre processus, et nous proposons donc aussi une version asynchrone de l'algorithme d'exclusion mutuelle.

Un *canal asynchrone* sera modélisé par une variable ordinaire C. La transmission de la valeur e sur ce canal est modélisée par l'affectation $C := e$, tandis que son extraction, dans une variable x, sera modélisée par l'affectation conditionnelle $C \neq \emptyset \longrightarrow (x, C) := (C, \emptyset)$. (Le symbole $\emptyset$ désigne ici un état particulier du canal, correspondant à l'absence d'information utile.)
Dans le cas particulier d'un canal de synchronisation pure, la transmission et la réception seront modélisées respectivement par les instructions $C := 1$ et $C = 1 \longrightarrow C := 0$.

La réécriture sous forme asynchrone de notre système est immédiate (voir figure 4.17).

Par contre, le fait que la version synchrone soit correcte n'implique pas l'exactitude de la version asynchrone; une nouvelle adaptation de l'invariant est nécessaire pour prouver ce dernier point. Un invariant adéquat de la version asynchrone est

$$
\begin{array}{ll}
 & \forall r\ [\,at\ r_w = (at\ C_r + REQ_r + OK_r + r \in E)\,] \\
\wedge & \forall r\ [\,(SC = r) = (at\ C'_r + OK_r + END_r + at\ r_c)\,] \\
\wedge & (SC = 0\ \Rightarrow\ E = \emptyset)\,.
\end{array}
$$

Il serait souhaitable de prouver que tout processus demandant l'accès à sa section critique finisse par l'obtenir. Une telle preuve est possible, mais subordonnée à une série d'hypothèses que nous énonçons informellement ci-dessous.

- Aucun processus ne reste indéfiniment en section critique.

$(p_0,\ \neg rcs_p \longrightarrow [\,],\ p_0)$,
$(p_0,\ rcs_p \longrightarrow REQ_p := 1\ ,\ p_w)$,
$(p_w,\ OK_p = 1 \longrightarrow OK_p := 0\ ,\ p_c)$,
$(p_c,\ rcs_p \longrightarrow [\,],\ p_c)$,
$(p_c,\ \neg rcs_p \longrightarrow END_p := 1\ ,\ p_0)$,

$(C,\ REQ_p = 1 \longrightarrow REQ_p := 0\ ,\ C_p)$,
$(C_p,\ SC = 0 \longrightarrow SC := p;\ OK_p := 1\ ,\ C)$,
$(C_p,\ SC \neq 0 \longrightarrow E := E \cup \{p\}\ ,\ C)$,
$(C,\ END_p = 1 \longrightarrow END_p := 0\ ,\ C'_p)$,
$(C'_p,\ q \in E \longrightarrow (SC, E) := (q, E \setminus \{q\});\ OK_q := 1\ ,\ C)$,
$(C'_p,\ E = \emptyset \longrightarrow SC := 0\ ,\ C)$.

Figure 4.17 *Transitions d'un système asynchrone d'exclusion mutuelle*

- Le système global ne reste indéfiniment inactif que quand aucune transition n'est exécutable.
- Le contrôleur ne reste indéfiniment inactif que quand aucune requête ou avis de fin n'attend un traitement.
- Entre le moment où un processus est prêt à émettre une requête ou un avis de fin, et celui où la communication est émise, seul un nombre fini de communications et d'opérations internes peuvent avoir lieu.
- Entre le moment où un processus entre dans l'ensemble d'attente et celui où il en sort, seul un nombre fini de processus (non nécessairement distincts) peuvent accéder à la section critique.

La logique classique est assez peu appropriée pour l'expression et la démonstration de telles assertions, dans lesquelles le temps joue un rôle important. La logique temporelle, qui sera introduite dans le volume III, permettra de traiter plus facilement ce type d'assertions.

4.6. Exclusion mutuelle à contrôle réparti

L'algorithme d'exclusion mutuelle que nous venons de présenter repose sur l'intervention d'un contrôleur. Il est possible, et parfois préférable, de ne pas recourir à un processus de contrôle qui, en cas de défaillance, anéantit le système entier. Nous introduisons dans ce paragraphe un algorithme d'exclusion mutuelle (Ricart et Agrawala, 1981) qui fonctionne sans contrôle centralisé. Tous les processus concourent au bon fonctionnement du système et au maintien de l'exclusion mutuelle.

4.6.1. *Description informelle*

Un réseau comporte $N \geq 2$ *stations*, ou *nœuds*, qui communiquent par échange de messages. Ils doivent assurer l'accès en exclusion mutuelle à une ressource partagée.

L'idée de base de Ricart et Agrawala est la suivante. Une station souhaitant entrer en section critique doit envoyer une requête à toutes les autres stations. Au reçu de la requête, chaque station décide sur base d'une fonction de priorité d'envoyer immédiatement l'autorisation, ou de postposer cet envoi jusqu'après son propre passage en section critique. Quand la station demandeuse a reçu les $N - 1$ autorisations, elle accède à sa section critique. Sa tâche achevée, la station envoie les autorisations correspondant aux requêtes pendantes éventuelles, puis retourne en section non critique.

Nous utilisons les notations introduites dans l'article de Ricart et Agrawala :

rcs_p : l'accès de p à la section critique est nécessaire;
RCS_p : l'accès de p à la section critique est demandé;
OR_p^q : p attend une réponse de q (Outstanding Reply);
RD_p^q : p retarde sa réponse à q (Reply Deferred);
PR_p^q : p a priorité sur q (PRiority).

Le comportement d'une station p est schématisé à la figure 4.18.

Remarque. L'idée de base de Ricart et Agrawala semble relativement simple et peut se résumer en quelques phrases. Néanmoins, il n'est pas évident que la solution proposée garantisse l'exclusion mutuelle en permanence, tout en ne provoquant aucun blocage. D'ailleurs, le schéma proposé à la figure 4.18 est à la fois trop vague et trop abstrait pour donner lieu immédiatement à un algorithme correct et efficace. Un tel algorithme sera obtenu par transformations successives, et chaque transformation sera soigneusement vérifiée avant toute transformation ultérieure.

4.6.2. *Une première version abstraite*

On obtient une version simplifiée au maximum en supprimant tout problème de communication et en ne retenant pour chaque station p que les points de contrôle essentiels : état non critique p_0, état d'attente p_4 et état critique p_5.

Chaque station p est comme précédemment modélisée par un processus de même nom. L'ensemble des processus est $\mathcal{P}_0 = \{p, q, r, \ldots\}$. Chaque processus contient trois étiquettes.

Pour chaque station p, le système $\mathcal{S}_0$ contient les transitions décrites à la figure 4.19. Comme précédemment, le symbole [ ] représente un pas d'exécution interne à la station p. Un tel pas peut modifier la variable rcs_p, mais est sans effet sur les autres variables et expressions introduites jusqu'ici.

La signification intuitive des transitions est la suivante.

$$
\begin{array}{rl}
\textit{Section non critique} & \neg rcs_p \longrightarrow [\,] \\
& p_0 \\
\text{Accès nécessaire} & rcs_p \longrightarrow RCS_p := 1 \\
& p_1 \\
\text{Basse priorité} & \forall q{\neq}p\,[(PR_p^q, PR_q^p) := (0,1)] \\
& p_2 \\
\text{Envoi des requêtes} & \forall q{\neq}p\,[OR_p^q := 1] \\
& p_3 \\
\text{Décisions} & \forall q{\neq}p\,[\text{if } RCS_q \wedge PR_q^p \text{ then } RD_q^p := 1 \text{ else } OR_p^q := 0] \\
& p_4 \\
\text{Attente des réponses} & \forall q{\neq}p\,[\neg OR_p^q] \longrightarrow \textit{skip} \\
& p_5 \\
\textit{Section critique} & rcs_p \longrightarrow [\,] \\
& p_5 \\
\text{Fin d'accès} & \neg rcs_p \longrightarrow RCS_p := 0 \\
& p_6 \\
\text{Réponses différées} & \forall q{\neq}p\,[\text{if } RD_p^q \text{ then } (RD_p^q, OR_q^p) := (0,0)] \\
& p_0 \\
\textit{Section non critique} & \neg rcs_p \longrightarrow [\,]
\end{array}
$$

Figure 4.18 *Station p*

$$
\begin{array}{l}
(p_0,\ \neg rcs_p \longrightarrow [\,],\ p_0)\ , \\
(p_0,\ rcs_p \longrightarrow RCS_p := 1;\ \forall q{\neq}p\,[(PR_p^q, PR_q^p, OR_p^q) := (0,1,1)]; \\
\qquad\qquad \forall q{\neq}p\,[\text{if } RCS_q \wedge PR_q^p \text{ then } RD_q^p := 1 \text{ else } OR_p^q := 0],\ p_4)\ , \\
(p_4,\ \forall q{\neq}p\,[\neg OR_p^q] \longrightarrow \textit{skip},\ p_5)\ , \\
(p_5,\ rcs_p \longrightarrow [\,],\ p_5)\ , \\
(p_5,\ \neg rcs_p \longrightarrow RCS_p := 0;\ \forall q{\neq}p\,[\text{if } RD_p^q \text{ then } (RD_p^q, OR_q^p) := (0,0)],\ p_0)\ .
\end{array}
$$

Figure 4.19 *Transitions du système $\mathcal{S}_0$*

- $(p_0 \to p_0)$ modélise l'exécution interne en section non critique.
- $(p_0 \to p_4)$ modélise le protocole d'entrée. La station p indique que l'accès à la section critique est nécessaire, "prend la file" en s'attribuant la plus faible priorité et envoie une requête à chaque autre station. Les autres stations réagissent instantanément, puisque dans cette version simplifiée, on ne considère pas le problème lié aux communications.
- $(p_4 \to p_5)$ modélise l'attente précédant l'accès.
- $(p_5 \to p_5)$ modélise l'exécution interne en section non critique.
- $(p_5 \to p_0)$ modélise le protocole de sortie. La station p indique que l'accès à la section critique n'est plus nécessaire, et envoie des réponses

aux autres stations ayant demandé l'accès (les réactions à ces réponses sont aussi instantanées).

Cette version simplifiée à l'extrême n'est naturellement pas satisfaisante; nous analysons maintenant ses défectuosités.

- Les communications d'une station p à une station q sont modélisées par des affectations exécutées soit par p, soit par q. Ceci n'est pas réaliste: les deux stations doivent coopérer pour échanger de l'information.
- Comme la fonction de priorité est évaluée immédiatement après avoir été adaptée par une station demandeuse, c'est-à-dire avant toute interférence d'autres stations, l'expression $(RCS_q \wedge PR_q^p)$ se réduira toujours à RCS_q dans la transition $(p_0 \rightarrow p_4)$.
- De grandes parties du programme ont été artificiellement traitées comme atomiques et modélisées par une seule transition, alors qu'elles comportent plusieurs actions parfois complexes. En particulier, il est irréaliste de supposer qu'une station puisse émettre et/ou recevoir plusieurs messages à la fois.

Dans la suite de ce chapitre, ces suppositions seront remplacées par d'autres, plus réalistes. Néanmoins, nous allons d'abord formaliser le comportement du système simplifié $\mathcal{S}_0$, en construisant un invariant. Cet invariant doit être suffisamment fort pour établir les spécifications du système.

La spécification principale est évidemment l'exclusion mutuelle, formalisée par l'assertion

(4.3) $\quad \forall p \forall q \neq p \, [\neg(at\ p_5 \wedge at\ q_5)]$.

L'assertion

(4.4) $\quad \forall p \forall q \neq p \, [(at\ p_4 \wedge at\ q_4) \Rightarrow OR_p^q + OR_q^p = RD_p^q + RD_q^p = 1]$

formalise le concept de réponse immédiate ou différée.

Il ne sera pas difficile d'obtenir un invariant adéquat pour $\mathcal{S}_0$ ... puisque toutes les difficultés ont été (provisoirement) escamotées. L'invariant ne sera rien d'autre que la définition axiomatique des variables et expressions déjà introduites. Nous les passons en revue.

- Le cas de RCS_p est très simple, puisque cette variable n'est altérée que par la station p. L'assertion adéquate est $at\ p_0 \equiv \neg RCS_p$.
- A chaque instant (y compris à l'instant initial), exactement une des deux expressions PR_p^q et PR_q^p est vraie, ce que nous exprimons par l'assertion $PR_p^q + PR_q^p = 1$. En outre, si une station p a obtenu l'accès alors qu'une autre station q le demandait, c'est que la première avait priorité sur la seconde. Cela est exprimé par l'assertion $(at\ p_5 \wedge RCS_q) \Rightarrow PR_p^q$.
- Le rôle des expressions OR_p^q, OR_q^p, RD_p^q et RD_q^p est la résolution des conflits entre les stations p et q. La station p attend une réponse de la station q si et seulement si la station q a différé une réponse à la station p;

on a donc $OR_p^q \equiv RD_q^p$. De plus, en cas de réponse retardée, la station p est nécessairement dans son état d'attente p_4; enfin, puisque la station q a demandé l'accès, la variable RCS_q est vraie. Tout ceci est résumé en l'assertion $RD_q^p \equiv (at\, p_4 \wedge RCS_q \wedge PR_q^p)$.

L'invariant I_0 sera simplement la conjonction de toutes ces assertions, que nous récapitulons ci-dessous.

$$
\begin{array}{ll}
 & at\, p_0 \equiv \neg RCS_p\,, \\
 & PR_p^q + PR_q^p = 1\,, \\
(4.5) & (at\, p_5 \wedge RCS_q) \Rightarrow PR_p^q\,, \\
 & OR_p^q \equiv RD_q^p\,, \\
 & RD_q^p \equiv (at\, p_4 \wedge RCS_q \wedge PR_q^p)\,.
\end{array}
$$

Tout état vérifiant l'invariant est a priori un état initial acceptable. Habituellement, on partira d'un état dans lequel toutes les stations sont au repos; on aura

$$at\, p_0 \wedge \neg RCS_p$$

pour toute station p et, si q est une autre station, on aura en outre

$$(PR_p^q \equiv p < q) \wedge \neg OR_p^q \wedge \neg OR_q^p \wedge \neg RD_p^q \wedge \neg RD_q^p\,,$$

où $<$ représente un ordre total strict (arbitraire) entre les stations.

On vérifie aisément que I_0 est un invariant de $\mathcal{S}_0$, c'est-à-dire que le triplet $\{I_0\}\,\tau\,\{I_0\}$ est vrai pour toute transition τ de $\mathcal{S}_0$. On observe aussi que l'exclusion mutuelle est respectée : si $at\, p_5$ et $at\, q_5$ étaient vrais simultanément, on aurait, au même instant, RCS_p, RCS_q, PR_p^q et PR_q^p, ce qui contredirait l'invariant dans son assertion $PR_p^q + PR_q^p = 1$.

Remarque. L'avantage d'une version aussi dépouillée d'un algorithme est la mise en évidence de l'idée sous-jacente. On va maintenant, par une succession de raffinements, rendre le programme plus réaliste — c'est-à-dire réalisable — en préservant le principe et les propriétés essentielles de ce programme. (Sur les diverses notions de raffinement, on consultera notamment Back (1983, 1988, 1989), Morris (1987) et Morgan (1988).)

4.6.3. *Un grain plus fin*

Pour chaque station p, la transition

$$
\begin{array}{l}
(p_0,\ rcs_p \longrightarrow RCS_p := 1;\ \forall\, q \neq p\,[(PR_p^q, PR_q^p, OR_p^q) := (0,1,1)]; \\
\qquad\qquad \forall\, q \neq p\,[\text{if } RCS_q \wedge PR_q^p \text{ then } RD_q^p := 1 \text{ else } OR_p^q := 0]\,,\ p_4)
\end{array}
$$

comporte trop d'actions et doit être décomposée. La première ligne représente la phase préliminaire, pendant laquelle la station demandeuse manifeste son intention et envoie ses requêtes. La seconde ligne modélise la phase de décision : chaque station q distincte de p doit décider si elle a ou non priorité sur p. Nous allons séparer les deux phases, en introduisant une étiquette intermédiaire p_3. On remplace donc la transition $(p_0 \rightarrow p_4)$ par les transitions

$(p_0,\ rcs_p \longrightarrow RCS_p := 1;\ \forall q \neq r\ [(PR_p^q, PR_q^p, OR_p^q) := (0,1,1)]\ ,\ p_3)\ ,$
$(p_3,\ \forall q \neq r\ [\text{if } RCS_q \wedge PR_q^p \text{ then } RD_q^p := 1 \text{ else } OR_p^q := 0]\ ,\ p_4)\ .$

On a obtenu un système $\mathcal{S}_1$; pour le valider, il faut montrer que l'invariant I_0 ne doit pas être modifié (cas idéal) ou du moins que les modifications nécessaires sont acceptables. Plus précisément, l'assertion exprimant l'exclusion mutuelle doit rester une conséquence logique de l'invariant modifié.

Le cas présent n'est pas idéal, car les assertions du type $OR_p^q \equiv RD_q^p$ ne sont pas respectées. Plus précisément, dans l'état intermédiaire p_3, OR_p^q est vrai et RD_q^p est faux, et on ne peut pas choisir $I_1 = I_0$.

Le remède est simple; il suffit de remplacer les assertions du type

$$OR_p^q \equiv RD_q^p$$

par

$$(RD_q^p \Rightarrow OR_p^q) \wedge (\neg at\ p_3 \equiv (OR_p^q \Rightarrow RD_q^p))\ .$$

Ceci capture exactement le changement concernant l'équivalence $OR_p^q \equiv RD_q^p$. D'une part, l'implication inverse reste vraie et, d'autre part, l'implication directe devient fausse si et seulement si on se trouve au point de contrôle intermédiaire p_3. Ce changement est acceptable, car l'assertion exprimant l'exclusion mutuelle reste une conséquence logique de l'invariant modifié.

Remarque. Il est naturellement nécessaire de montrer que l'invariant modifié est bien un invariant. Cette vérification de routine est laissée au lecteur; nous nous contenterons dans ce chapitre d'une vérification générale, concernant la dernière version que nous obtiendrons.

Insistons sur le fait qu'une vérification facile n'est pas une vérification inutile. Certaines décompositions, quoique naturelles intuitivement, ne sont pas valides. Une décomposition de ce type est suggérée à la figure 4.18, qui comporte une étiquette p_1. Pour toute station p, la transition

$(p_0,\ rcs_p \longrightarrow RCS_p := 1;\ \forall q \neq p\ [(PR_p^q, PR_q^p, OR_p^q) := (0,1,1)]\ ,\ p_3)$

pourrait être remplacée, dans la variante $\mathcal{S}_2$, par les deux transitions

$(p_0,\ rcs_p \longrightarrow RCS_p := 1\ ,\ p_1)\ ,$
$(p_1,\ \forall q \neq p\ [(PR_p^q, PR_q^p, OR_p^q) := (0,1,1)]\ ,\ p_3)\ .$

Il se fait malheureusement que cette décomposition altère fortement l'invariant; la plupart des assertions originelles sont falsifiées, et il n'est pas évident de les remplacer. Plusieurs tentatives vaines doivent inciter le programmeur à la méfiance. Plus concrètement, la décomposition en question n'aurait-elle pas pour effet de permettre des exécutions inacceptables? C'est le cas ici, et un exemple d'exécution inacceptable est donnée à la figure 4.20. L'exécution en question respecte l'exclusion mutuelle, mais *les deux* stations p et q sont mises en attente. Dans le dernier état de cette exécution, les stations sont bloquées définitivement dans leur état d'attente. Cet interblocage fatal porte en anglais le nom imagé de *deadlock*.

p	q	OR_p^q	OR_q^p	RD_p^q	RD_q^p	PR_p^q	PR_q^p
p_0	q_0	0	0	0	0	*	*
p_1	q_0	0	0	0	0	*	*
p_1	q_3	0	1	0	0	1	0
p_1	q_4	0	1	1	0	1	0
p_3	q_4	1	1	1	0	0	1
p_4	q_4	1	1	1	1	0	1

Figure 4.20 *Exécution conduisant à un interblocage*

Le système $\mathcal{S}_2$ n'est pas acceptable et nous devons donc renoncer, par rapport au schéma de la figure 4.18, à l'introduction de l'étiquette p_1.[6] Par contre, il est licite d'introduire non seulement p_3, comme nous venons de le voir, mais aussi p_2 et p_6. On obtient alors le système $\mathcal{S}_3$, dont les transitions sont, pour chaque station p,

$$
\begin{array}{l}
(p_0,\ \neg rcs_p \longrightarrow [\,],\ p_0)\,,\\
(p_0,\ rcs_p \longrightarrow RCS_p := 1\,;\ \forall q{\neq}p\,[(PR_p^q, PR_q^p) := (0,1)],\ p_2)\,,\\
(p_2,\ \forall q{\neq}p\,[OR_p^q := 1],\ p_3)\,,\\
(p_3,\ \forall q{\neq}p\,[\text{if } RCS_q \wedge PR_q^p \text{ then } RD_q^p := 1 \text{ else } OR_p^q := 0],\ p_4)\,,\\
(p_4,\ \forall q{\neq}p\,[\neg OR_p^q] \longrightarrow skip,\ p_5)\,,\\
(p_5,\ rcs_p \longrightarrow [\,],\ p_5)\,,\\
(p_5,\ \neg rcs_p \longrightarrow RCS_p := 0\,,\ p_6)\,,\\
(p_6,\ \forall q{\neq}p\,[\text{if } RD_p^q \text{ then } (RD_p^q, OR_q^p) := (0,0)],\ p_0)\,.
\end{array}
\tag{4.6}
$$

L'invariant correspondant I_3 est

$$
\begin{array}{l}
\forall p\,[at\ p_{06} \equiv \neg RCS_p]\,,\\
\forall p \forall q{\neq}p\,[PR_p^q + PR_q^p = 1 \wedge ((at\ p_5 \wedge RCS_q) \Rightarrow PR_p^q)]\,,\\
\forall p \forall q{\neq}p\,[(RD_q^p \Rightarrow OR_p^q) \wedge (\neg at\ p_3 \equiv (OR_p^q \Rightarrow RD_q^p))]\,,\\
\forall p \forall q{\neq}p\,[at\ p_6 q_4 \vee (RD_p^q \equiv (at\ q_4 \wedge RCS_p \wedge PR_p^q))]\,.
\end{array}
\tag{4.7}
$$

Remarque. La notation $at\ p_{06}$ abrège la formule $at\ p_0 \vee at\ p_6$.

Les conséquences fâcheuses qu'aurait l'introduction des étiquettes intermédiaires de type p_1 doivent nous inciter à vérifier formellement l'absence d'interblocage, mais nous établirons cette propriété à propos de la version définitive seulement.

Il est assez clair, en effet, que le système $\mathcal{S}_3$ n'est pas encore satisfaisant. Les opérations distinctes sont maintenant modélisées par des transitions distinctes, mais certaines transitions comportent encore $N-1$ opérations analogues.

[6] Il n'est pas sans intérêt de remarquer que la première version publiée de cet algorithme comportait l'erreur qui vient d'être évoquée ici. Ceci montre que la prudence est indispensable en algorithmique, spécialement lorsqu'il y a parallélisme.

4.6.4. *Réalisation du schéma de priorité*

Comme nous l'avons mentionné dans le paragraphe 4.6.1, le schéma de priorité est simplement "premier entré, premier sorti". La manière très abstraite dont nous l'avons modélisé convient bien pour la spécification et la preuve de propriétés, mais, pour la réalisation, il est souhaitable d'éviter le partage de variables. Dans un premier temps, on voudrait au moins limiter l'accès en écriture. Chaque variable serait lue par toutes les stations, mais modifiée par une seule. Nous utiliserons pour ce faire une technique bien connue, dite "de la boulangerie" (Lamport, 1974). Le principe est simple. Dans une boulangerie, chaque client prend un ticket numéroté en entrant, et les clients sont servis dans l'ordre des numéros.

Cette technique s'adapte immédiatement à notre application. Chaque station enregistre le temps auquel elle annonce sa demande d'accès. La station p aura priorité sur la station q si elle a enregistré un temps antérieur, c'est-à-dire si $SN_p < SN_q$. Le symbole SN signifie "Sequence Number".

On modifie les transitions relatives à la station p comme suit :

$$
\begin{array}{l}
(p_0,\ \neg rcs_p \longrightarrow [\,],\ p_0)\,, \\
(p_0,\ rcs_p \longrightarrow (RCS_p, SN_p) := (1, time),\ p_2)\,, \\
(p_2,\ \forall q{\neq}p\,[OR_p^q := 1],\ p_3)\,, \\
(p_3,\ \forall q{\neq}p\,[\text{if } RCS_q \wedge SN_q < SN_p \text{ then } RD_q^p := 1 \text{ else } OR_p^q := 0],\ p_4)\,, \\
(p_4,\ \forall q{\neq}p\,[\neg OR_p^q] \longrightarrow skip,\ p_5)\,, \\
(p_5,\ rcs_p \longrightarrow [\,],\ p_5)\,, \\
(p_5,\ \neg rcs_p \longrightarrow RCS_p := 0,\ p_6)\,, \\
(p_6,\ \forall q{\neq}p\,[\text{if } RD_p^q \text{ then } (RD_p^q, OR_q^p) := (0,0)],\ p_0)\,.
\end{array}
\tag{4.8}
$$

La modification conduit au système $\mathcal{S}_4$; elle doit se justifier par un invariant I_4, ce qui, dans le cas présent, est très simple : il suffit de remplacer dans l'invariant I_3 chaque occurrence de PR_p^q par $SN_p < SN_q$, ce qui donne

$$
\begin{array}{l}
\forall p\,[at\ p_{06} \equiv \neg RCS_p]\,, \\
\forall p \forall q{\neq}p\,[(SN_p{\neq}SN_q) \wedge ((at\ p_5 \wedge RCS_q) \Rightarrow SN_p < SN_q)]\,, \\
\forall p \forall q{\neq}p\,[(RD_q^p \Rightarrow OR_p^q) \wedge (\neg at\ p_3 \equiv (OR_p^q \Rightarrow RD_q^p))]\,, \\
\forall p \forall q{\neq}p\,[at\ p_6 q_4 \vee (RD_p^q \equiv (at\ q_4 \wedge RCS_p \wedge SN_p < SN_q))]\,.
\end{array}
\tag{4.9}
$$

Remarque. On suppose qu'il est impossible pour deux stations d'enregistrer exactement le même temps, d'où l'assertion $SN_p \neq SN_q$.

4.6.5. *Décomposition fine des transitions*

Certaines transitions comportent des affectations ou des gardes quantifiées; celles concernant la station p sont $(p_2 \to p_3)$, $(p_3 \to p_4)$, $(p_4 \to p_5)$ et $(p_6 \to p_0)$. Ces transitions impliquent une communication "générale" entre la station p et les $N-1$ autres stations, et il serait irréaliste de considérer ce type de

communication comme atomique. Chacune de ces transitions doit donc être décomposée en $N-1$ transitions distinctes, correspondant à une communication entre deux stations seulement.

Cela pourrait se faire en introduisant, pour chaque transition à décomposer, $N-2$ points de contrôle supplémentaires. Outre le fait qu'une telle solution serait lourde et inélégante, elle aurait l'inconvénient d'imposer l'ordre dans lequel la station p communique avec les autres stations. Pour éviter ces inconvénients, on introduira plutôt une variable supplémentaire X_p, qui mémorisera l'ensemble des stations avec lesquelles p a déjà communiqué. Initialement, cette variable de type ensemble sera vide; à chaque communication, le nom de la station avec laquelle p communique est ajouté à X_p, jusqu'à ce que la valeur de cette variable soit $\mathcal{P}_p =_{def} \mathcal{P}_0 \backslash \{p\}$.

Concrètement, la transition

$$(p_2,\ \forall q \in \mathcal{P}_p\ [OR_p^q := 1]\ ,\ p_3)$$

sera remplacée par les transitions

$$(p_2,\ q \in \mathcal{P}_p \backslash X_p \longrightarrow (OR_p^q, X_p) := (1, X_p \cup \{q\}),\ p_2)\ ,$$
$$(p_2,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_3)\ .$$

Une fois de plus, il est indispensable de valider cette transformation par une adaptation de l'invariant. Comme d'habitude, il suffira de formaliser l'idée intuitive sous-jacente. Avant la transformation, le prédicat $at\ p_2$ était vrai avant que p ait communiqué avec une autre station q, alors que $at\ p_3$ était vrai après la communication. La conséquence de la transformation est la modification de ces prédicats. Avant la communication entre p et q, le prédicat $(at\ p_2 \wedge q \notin X_p)$ est vrai; après cette communication, le prédicat $[at\ r_3 \vee (at\ r_2 \wedge q \in X_p)]$ est vrai.

Le cas des transitions $(p_3 \rightarrow p_4)$, $(p_4 \rightarrow p_5)$ et $(p_6 \rightarrow p_0)$ est tout à fait analogue. Leur décomposition nous amène au système $\mathcal{S}_5$, dont l'ensemble de transitions est donné à la figure 4.21.

Le nouvel invariant est obtenu à partir de l'ancien en ajoutant l'assertion $\forall p\,(X_p \subset \mathcal{P}_p)$ et en remplaçant toute occurrence du prédicat $at\ p_i$ survenant dans la portée du quantificateur $\forall p \neq q$ par le prédicat $at^q\ p_i$; la signification intuitive de ce nouveau prédicat est simple: "du point de vue de la station q, la station p se trouve au point de contrôle p_i". Les définitions formelles des nouveaux prédicats sont récapitulées à la figure 4.22.
L'invariant I_5 est la conjonction, pour toutes les stations distinctes p, q, des assertions représentées à la figure 4.23.

4.7. Vers un système réel

Le système représenté à la figure 4.21 peut être réalisé par un ensemble de processus communiquant par variables partagées. Cependant, comme les expressions sont partagées par au plus deux stations, on peut envisager un

$$
\begin{array}{ll}
\tau^{p}_{00}: & (p_0,\ \neg rcs_p \longrightarrow [\,],\ p_0)\,,\\
\tau^{p}_{02}: & (p_0,\ rcs_p \longrightarrow (RCS_p, SN_p) := (1, time),\ p_2)\,,\\
\tau^{pq}_{22}: & (p_2,\ q \in \mathcal{P}_p \backslash X_p \longrightarrow (OR^q_p, X_p) := (1, X_p \cup \{q\}),\ p_2)\,,\\
\tau^{p}_{23}: & (p_2,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_3)\,,\\
\tau^{pq}_{33a}: & (p_3,\ q \in \mathcal{P}_p \backslash X_p \wedge RCS_q \wedge SN_q < SN_p \longrightarrow\\
 & \quad (RD^p_q, X_p) := (1, X_p \cup \{q\}),\ p_3)\,,\\
\tau^{pq}_{33b}: & (p_3,\ q \in \mathcal{P}_p \backslash X_p \wedge [\neg RCS_q \vee SN_p < SN_q] \longrightarrow\\
 & \quad (OR^q_p, X_p) := (0, X_p \cup \{q\}),\ p_3)\,,\\
\tau^{p}_{34}: & (p_3,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_4)\,,\\
\tau^{pq}_{44}: & (p_4,\ q \in \mathcal{P}_p \backslash X_p \wedge \neg OR^q_p \longrightarrow X_p := X_p \cup \{q\},\ p_4)\,,\\
\tau^{p}_{45}: & (p_4,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_5)\,,\\
\tau^{p}_{55}: & (p_5,\ rcs_p \longrightarrow [\,],\ p_5)\,,\\
\tau^{p}_{56}: & (p_5,\ \neg rcs_p \longrightarrow RCS_p := 0,\ p_6)\,,\\
\tau^{pq}_{66a}: & (p_6,\ q \in \mathcal{P}_p \backslash X_p \wedge RD^q_p \longrightarrow\\
 & \quad (RD^q_p, OR^p_q, X_p) := (0, 0, X_p \cup \{q\}),\ p_6)\,,\\
\tau^{pq}_{66b}: & (p_6,\ q \in \mathcal{P}_p \backslash X_p \wedge \neg RD^q_p \longrightarrow X_p := X_p \cup \{q\},\ p_6)\,,\\
\tau^{p}_{60}: & (p_6,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_0)\,.
\end{array}
$$

Figure 4.21 *Décomposition des communications générales*

$$
\begin{array}{ll}
at^q\, p_0 & =_{def}\ ((at\ p_6 \wedge q \in X_p) \vee at\ p_0)\,,\\
at^q\, p_2 & =_{def}\ (at\ p_2 \wedge q \in \mathcal{P}_p \backslash X_p)\,,\\
at^q\, p_3 & =_{def}\ ((at\ p_2 \wedge q \in X_p) \vee (at\ p_3 \wedge q \in \mathcal{P}_p \backslash X_p))\,,\\
at^q\, p_4 & =_{def}\ ((at\ p_3 \wedge q \in X_p) \vee (at\ p_4 \wedge q \in \mathcal{P}_p \backslash X_p))\,,\\
at^q\, p_5 & =_{def}\ ((at\ p_4 \wedge q \in X_p) \vee at\ p_5)\,,\\
at^q\, p_6 & =_{def}\ (at\ p_6 \wedge q \in \mathcal{P}_p \backslash X_p)\,.
\end{array}
$$

Figure 4.22 *Définitions des nouveaux prédicats de contrôle*

$$
\begin{array}{ll}
1_p: & [X_p \subset \mathcal{P}_p \wedge (at\ p_{05} \Rightarrow X_p = \emptyset)]\,,\\
2_p: & [at\ p_{06} \equiv \neg RCS_p]\,,\\
3_{pq}: & [(SN_p \neq SN_q) \wedge ((at^q\, p_5 \wedge RCS_q) \Rightarrow SN_p < SN_q)]\,,\\
4_{pq}: & [(RD^p_q \Rightarrow OR^q_p) \wedge (\neg at^q\, p_3 \equiv (OR^q_p \Rightarrow RD^p_q))]\,,\\
5_{pq}: & [(at^q\, p_6 \wedge at^p\, q_4) \vee (RD^q_p \equiv (at^p\, q_4 \wedge RCS_p \wedge SN_p < SN_q))]\,.
\end{array}
$$

Figure 4.23 *Invariant après décomposition des communications*

ensemble de processus communiquant par échange de messages, ce qui est généralement plus efficace si les processus sont physiquement distants.

4.7.1. *Variables privées et transitions partagées*

Si on souhaite supprimer le partage de variables, il faut décider pour chaque variable quel processus a la charge de la modifier. On adopte la convention suivante, conforme à la signification intuitive des variables. Une variable est attachée à la station p si son indice *inférieur* est p.

Cette convention implique une nouvelle transformation du système. Considérons par exemple la transition

$$(p_6,\ q \in \mathcal{P}_p \backslash X_p \ \wedge\ RD_p^q \longrightarrow (RD_p^q, OR_q^p, X_p) := (0, 0, X_p \cup \{q\})\ ,\ p_6)\,.$$

Cette transition ne respecte pas la convention, puisqu'elle évoque et altère la variable OR_q^p, attachée à la station q. On peut remédier à cela assez facilement. Si le processus p n'est plus autorisé à accéder à la variable OR_q^p, un nouveau processus, dépendant de la station q, peut être créé dans ce but. En effet, on observe que la station p requiert la coopération de la station q de deux manières : d'abord quand l'accès est demandé (la station q doit alors connaître la valeur de SN_p), et ensuite quand une réponse est envoyée (la station q doit alors provoquer la remise à 0 de la variable OR_q^p).

Il est commode d'introduire deux nouveaux processus pour chaque station. Le premier recevra les requêtes émanant des autres stations et le second recevra les réponses. Pour la station p, les deux processus auxiliaires sont respectivement $p' = \{p'_0\}$ and $p'' = \{p''_0\}$.[7] Une station comporte maintenant trois processus, qui communiquent entre eux par partage de variables. Rappelons que le partage de variables n'est gênant qu'entre processus distants; ce n'est pas le cas de processus attachés à une même station.

Pour la station p, les transitions impliquant les processus auxiliaires sont

$$\begin{aligned}
&(p_3,\ q \in \mathcal{P}_p \backslash X_p \ \wedge\ RCS_q \ \wedge\ SN_q < SN_p \\
&\qquad\qquad \longrightarrow (RD_q^p, X_p) := (1, X_p \cup \{q\})\ ,\ p_3)\,, \\
&(p_3,\ q \in \mathcal{P}_p \backslash X_p \ \wedge\ (\neg RCS_q \ \vee\ SN_p < SN_q) \\
&\qquad\qquad \longrightarrow (OR_p^q, X_p) := (0, X_p \cup \{q\})\ ,\ p_3)\,, \\
&(p_6,\ q \in \mathcal{P}_p \backslash X_p \ \wedge\ RD_p^q \longrightarrow (RD_p^q, OR_q^p, X_p) := (0, 0, X_p \cup \{q\})\ ,\ p_6)\,.
\end{aligned}$$

La première évoque à la fois SN_q et RD_q^p; elle implique donc le premier processus auxiliaire attaché à la station q. La troisième transition évoque OR_q^p, qui doit être remise à 0 (faux) par le second processus auxiliaire attaché à la station q. Enfin, la deuxième transition implique non seulement le processus q' mais aussi le processus p'' (remise à 0 de OR_p^q).

Pour tenir compte de l'intervention des processus auxiliaires, les trois transitions sont récrites comme suit :

[7] Le symbole p désigne simultanément une station et le processus principal de cette station, chargé d'émettre les requêtes et les réponses. Cette double signification n'entraîne pas de confusion.

$$(p_3 q_0',\ q \in \mathcal{P}_p \backslash X_p \wedge RCS_q \wedge SN_q < SN_p \longrightarrow (RD_q^p, X_p) := (1, X_p \cup \{q\}),\ p_3 q_0'),$$

$$(p_3 q_0' p_0'',\ q \in \mathcal{P}_p \backslash X_p \wedge (\neg RCS_q \vee SN_p < SN_q) \longrightarrow (OR_p^q, X_p) := (0, X_p \cup \{q\}),\ p_3 q_0' p_0''),$$

$$(p_6 q_0'',\ q \in \mathcal{P}_p \backslash X_p \wedge RD_p^q \longrightarrow (RD_q^p, OR_q^p, X_p) := (0, 0, X_p \cup \{q\}),\ p_6 q_0'').$$

Cette transformation ne donne lieu à aucune modification de l'invariant. En effet, le changement ne concerne pas les *actions* effectuées, mais seulement les *acteurs* correspondants. La nouvelle version $\mathcal{S}_6$ du programme admet donc l'invariant $I_6 =_{def} I_5$. Elle n'implique plus de partage de variables, mais seulement la collaboration de processus. Une variable privée de la station p n'est altérée que par une transition impliquant l'un des processus attachés à la station p.

Intuitivement, les processus auxiliaires ont été introduits pour exécuter des tâches bien spécifiques. Soulignons cependant que, formellement, les processus du langage SFC ne sont que des ensembles de points de contrôle. Le concept d'agent, ou d'acteur, lié à un processus, ne fait pas partie de la sémantique du langage SFC, mais seulement de la pragmatique de son utilisation.

4.7.2. *Une horloge distribuée*

Il subsiste encore une entité partagée entre toutes les stations, à savoir le temps absolu, désigné par le symbole *time*. Il est sans doute optimiste de supposer que toutes les stations ont accès à une base de temps commune; en pratique, chaque station dispose de sa propre horloge, qui constitue une approximation plus ou moins fiable de la notion abstraite de temps absolu. Dans ce paragraphe, nous allons étudier dans quelle mesure le remplacement d'un temps global absolu par des copies approximatives et distribuées est acceptable.

La première étape consiste à expliciter et à formaliser ce que nous avons inconsciemment admis à propos de *time*. L'affectation

$$SN_p := time$$

était censée réaliser l'instruction plus abstraite

$$\forall\, q \neq p\ [(PR_p^q, PR_q^p) := (0, 1)],$$

étant donné l'équivalence $PR_p^q \equiv SN_p < SN_q$. En conséquence, nous avons admis la validité des triplets de Hoare

$$\{true\}\ SN_p := time\ \{SN_q < SN_p\},$$

pour toutes stations distinctes p et q. Ces triplets formalisent la *monotonie* du temps : il s'écoule à sens unique. Dans la mesure où des horloges distinctes ne sont pas nécessairement en parfaite concordance, l'hypothèse de monotonie globale ne pourra plus être admise sans restriction. Par contre, on peut

supposer que chaque horloge locale est monotone, ce qui revient à accepter la validité des triplets du type

$$\{SN_q < SN_p\}\ SN_p := time(p)\ \{SN_q < SN_p\},$$

alors que la validité des triplets du type

$$\{SN_q > SN_p\}\ SN_p := time(p)\ \{SN_q < SN_p\}$$

est sujette à caution. Heureusement, ces derniers triplets ne sont requis qu'au moment où une station q est sur le point d'accéder à sa propre section critique. Dans ce cas, la station p ne peut absolument pas être autorisée à "voler" la priorité en choisissant SN_p tel que $SN_p < SN_q$. Il faut donc synchroniser les horloges de p et q avant le moment où la station q atteint le point de contrôle $at^p q_4$, mais après le moment où cette station a utilisé sa propre horloge et adapté la variable SN_q. D'un point de vue opérationnel, c'est possible puisqu'il y a une communication entre les stations p et q dans cet intervalle de temps.

Le schéma de temps distribué choisi par Ricart et Agrawala est basé sur le "principe de la boulangerie". Chaque station p utilise une variable HSN_p qui contient le "numéro de ticket" le plus récent dont elle a eu connaissance, autrement dit, la valeur maximale des variables SN_q qui lui ont été communiquées. Quand la station p tente d'accéder à sa section critique, elle commence par prendre la priorité apparemment la plus basse, en choisissant pour SN_p une valeur supérieure à HSN_p, par exemple en exécutant l'affectation $SN_p := HSN_p + 1$; les autres stations adaptent leur copie locale du temps absolu en exécutant l'affectation $HSN_q := max(HSN_q, SN_p)$.

Pour permettre l'égalité accidentelle entre les valeurs de SN_p et SN_q, on supposera qu'un ordre total arbitraire a été déterminé entre les stations, qui résoudra le conflit en cas d'égalité. Le test $SN_p < SN_q$ est donc remplacé par le test $(SN_p, p) \prec (SN_q, q)$, où $\prec$ représente l'ordre lexicographique sur l'ensemble des couples. L'expression $(SN_p, p) \prec (SN_q, q)$ ou, pour abréger, $SN_p \prec SN_q$, est équivalente à l'expression $SN_p < SN_q \vee (SN_p = SN_q \wedge p < q)$.

Nous résumons les hypothèses faites à propos du temps distribué en la famille de triplets représentée ci-dessous.

(4.10) $\quad T_{pq} \ :\ \{SN_q \prec SN_p \vee at^p q_{45}\}\ (p_0 \rightarrow p_2)\ \{SN_q \prec SN_p\}.$

La validité de ces triplets est une conséquence immédiate d'un invariant spécial que nous devons ajouter à l'invariant du système; si p et q sont des stations distinctes, l'assertion C_{pq} est vraie.

(4.11) $\quad C_{pq} =_{def} [SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)].$

On obtient ainsi une nouvelle version $\mathcal{S}_7$ du système, dont les transitions sont représentées à la figure 4.24.

L'invariant I_7 s'obtient en remplaçant dans I_6 la relation $SN_p < SN_q$ par la relation $SN_p \prec SN_q$, et en ajoutant les assertions C_{pq}.

Si les horloges locales sont parfaites, l'accès à la section critique est accordé selon la règle "premier entré, premier servi".

$(p_0,\ \neg rcs_p \longrightarrow [\,],\ p_0)$,

$(p_0,\ rcs_p \longrightarrow (RCS_p, SN_p) := (1, HSN_p + 1),\ p_2)$,

$(p_2,\ q \in \mathcal{P}_p \backslash X_p \longrightarrow (OR_p^q, X_p) := (1, X_p \cup \{q\}),\ p_2)$,

$(p_2,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_3)$,

$(p_3 q_0',\ q \in \mathcal{P}_p \backslash X_p \wedge RCS_q \wedge SN_q \prec SN_p \longrightarrow$
$(RD_q^p, HSN_q, X_p) := (1, max(HSN_q, SN_p), X_p \cup \{q\}),\ p_3 q_0')$,

$(p_3 q_0' p_0'',\ q \in \mathcal{P}_p \backslash X_p \wedge [\neg RCS_q \vee SN_p \prec SN_q] \longrightarrow$
$(OR_p^q, HSN_q, X_p) := (0, max(HSN_q, SN_p), X_p \cup \{q\}),\ p_3 q_0' p_0'')$,

$(p_3,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_4)$,

$(p_4,\ q \in \mathcal{P}_p \backslash X_p \wedge \neg OR_p^q \longrightarrow X_p := X_p \cup \{q\},\ p_4)$,

$(p_4,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_5)$,

$(p_5,\ rcs_p \longrightarrow [\,],\ p_5)$,

$(p_5,\ \neg rcs_p \longrightarrow RCS_p := 0,\ p_6)$,

$(p_6 q_0'',\ q \in \mathcal{P}_p \backslash X_p \wedge RD_p^q \longrightarrow (RD_p^q, OR_q^p, X_p) := (0, 0, X_p \cup \{q\}),\ p_6 q_0'')$,

$(p_6,\ q \in \mathcal{P}_p \backslash X_p \wedge \neg RD_p^q \longrightarrow X_p := X_p \cup \{q\},\ p_6)$,

$(p_6,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_0)$.

Figure 4.24 *Transitions du système avec temps distribué*

4.7.3. *Evaluation locale des gardes*

Les transitions $(p_3 q_0' \rightarrow p_3 q_0')$ et $(p_3 q_0' p_0'' \rightarrow p_3 q_0' p_0'')$ comportent dans leur garde le test $SN_p \prec SN_q$. A priori, ce test ne sera pas effectué localement, puisque les variables à comparer appartiennent à des stations différentes. L'évaluation locale sera quand même possible si une station q maintient des copies locales des valeurs de type SN_p qui lui sont transmises. Les processus auxiliaires peuvent être utilisés dans ce but.

Concrètement, on introduit dans le premier processus auxiliaire de chaque station q une nouvelle étiquette, soit q_1' (on a maintenant $q' = \{q_0', q_1'\}$), et deux nouvelles variables k_q et ℓ_q. La signification intuitive du raffinement proposé est la suivante: quand *at* q_1' est vrai, la station q vient juste de recevoir une valeur SN_p, en provenance de la station p; la valeur et le nom de la station expéditrice sont respectivement rangés dans les nouvelles variables k_q et ℓ_q, ce qui permettra d'effectuer la comparaison entre SN_p et SN_q localement, en la station q.

Plus formellement, les transitions

$$(p_3 q_0',\ q \in \mathcal{P}_p \backslash X_p \wedge RCS_q \wedge SN_q \prec SN_p \longrightarrow (RD_q^p, HSN_q, X_p) := (1, max(HSN_q, SN_p), X_p \cup \{q\})\ ,\ p_3 q_0')\ ,$$

$$(p_3 q_0' p_0'',\ q \in \mathcal{P}_p \backslash X_p \wedge (\neg RCS_q \vee SN_p \prec SN_q) \longrightarrow (OR_p^q, X_p) := (0, X_p \cup \{q\})\ ,\ p_3 q_0' p_0'')$$

sont remplacées par les transitions

$$(p_3 q_0',\ q \in \mathcal{P}_p \backslash X_p \longrightarrow (k_q, \ell_q, X_p) := (SN_p, p, X_p \cup \{q\})\ ,\ p_3 q_1')\ ,$$

$$(q_1',\ RCS_q \wedge (SN_q, q) \prec (k_q, \ell_q) \longrightarrow (HSN_q, RD_q^{\ell_q}) := (max(HSN_q, k_q), 1)\ ,\ q_0')\ ,$$

$$(q_1' \ell_{q0}'',\ \neg RCS_q \vee (k_q, \ell_q) \prec (SN_q, q) \longrightarrow (HSN_q, OR_{\ell_q}^q) := (max(HSN_q, k_q), 1)\ ,\ q_0' \ell_{q0}'')\ .$$

Les nouvelles transitions $(q_1' \rightarrow q_0')$ et $(q_1' \ell_{q0}'' \rightarrow q_0' \ell_{q0}'')$ comportent encore plusieurs actions, ce qui suggère des décompositions supplémentaires. Une nouvelle variable booléenne d_q et deux nouvelles étiquettes q_2' et q_3' sont introduites pour chaque station q; les deux nouvelles transitions sont décomposées en

$$(q_1',\ HSN_q := max(HSN_q, k_q)\ ,\ q_2')\ ,$$
$$(q_2',\ d_q := (RCS_q \wedge (SN_q, q) \prec (k_q, \ell_q))\ ,\ q_3')\ ,$$
$$(q_3',\ d_q \longrightarrow RD_q^{\ell_q} := 1\ ,\ q_0')\ ,$$
$$(q_3' \ell_{q0}'',\ \neg d_q \longrightarrow OR_{\ell_q}^q := 0\ ,\ q_0' \ell_{q0}'')\ .$$

Une adaptation supplémentaire de l'invariant est nécessaire pour tenir compte de l'introduction des nouveaux objets. Comme ces objets (variables et étiquettes) sont locaux, l'adaptation est relativement simple et laissée au lecteur.

4.7.4. *Réalisation en pseudo-CSP*

L'élimination des variables partagées entre stations permet la réalisation du système sous forme d'un ensemble de processus de type CSP, le partage de variables ne subsistant qu'à l'échelon local. La transition

$$(p_3 q_0',\ q \in \mathcal{P}_p \backslash X_p \longrightarrow (k_q, \ell_q, X_p) := (SN_p, p, X_p \cup \{q\})\ ,\ p_3 q_1')$$

donne lieu aux pseudo-transitions

$$(p_3,\ q \in \mathcal{P}_p \backslash X_p \longrightarrow REQ_p^q!(SN_p, p);\ X_p := X_p \cup \{q\}\ ,\ p_3)\ ,$$
$$(q_0',\ REQ_p^q?(k_q, \ell_q) \longrightarrow skip\ ,\ q_1')\ .$$

Le symbole REQ_p^q désigne un canal de communication destiné à transmettre les requêtes de la station p vers la station q. L'instruction $REQ_p^q!(SN_p, p)$ correspond à l'émission de la requête, et l'instruction $REQ_p^q?(k_q, \ell_q)$ correspond à sa réception. On a de même un canal REP_p^q transportant les réponses de p vers q. Il s'agit d'un canal de synchronisation; les instructions correspondantes sont $REP_p^q?$ et $REP_p^q!$, sans argument.

Le processus principal attaché à la station p est responsable des transitions suivantes :

$$
\begin{aligned}
&(p_0,\ \neg rcs_p \longrightarrow [\,],\ p_0)\,,\\
&(p_0,\ rcs_p \longrightarrow (RCS_p, SN_p) := (1, HSN_p + 1),\ p_2)\,,\\
&(p_2,\ q \in \mathcal{P}_p \backslash X_p \longrightarrow (OR_p^q, X_p) := (1, X_p \cup \{q\}),\ p_2)\,,\\
&(p_2,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_3)\,,\\
&(p_3,\ q \in \mathcal{P}_p \backslash X_p \longrightarrow REQ_p^q!(SN_p, p);\ X_p := X_p \cup \{q\},\ p_3)\,,\\
&(p_3,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_4)\,,\\
&(p_4,\ q \in \mathcal{P}_p \backslash X_p \wedge \neg OR_p^q \longrightarrow X_p := X_p \cup \{q\},\ p_4)\,,\\
&(p_4,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_5)\,,\\
&(p_5,\ rcs_p \longrightarrow [\,],\ p_5)\,,\\
&(p_5,\ \neg rcs_p \longrightarrow RCS_p := 0,\ p_6)\,,\\
&(p_6,\ q \in \mathcal{P}_p \backslash X_p \wedge RD_p^q\\
&\qquad\qquad \longrightarrow REP_p^q!;\ (RD_p^q, X_p) := (0, X_p \cup \{q\}),\ p_6)\,,\\
&(p_6,\ q \in \mathcal{P}_p \backslash X_p \wedge \neg RD_p^q \longrightarrow X_p := X_p \cup \{q\},\ p_6)\,,\\
&(p_6,\ X_p = \mathcal{P}_p \longrightarrow X_p := \emptyset,\ p_0)\,.
\end{aligned}
\tag{4.12}
$$

Les transitions et pseudo-transitions impliquant le premier processus auxiliaire attaché à la station p sont

$$
\begin{aligned}
&(p_0',\ REQ_q^p?(k_p, \ell_p) \longrightarrow skip,\ p_1')\,,\\
&(p_1',\ HSN_p := max(HSN_p, k_p),\ p_2')\,,\\
&(p_2',\ d_p := (RCS_p \wedge (SN_p, p) \prec (k_p, \ell_p)),\ p_3')\,,\\
&(p_3',\ d_p \longrightarrow RD_p^{\ell_p} := 1,\ p_0')\,,\\
&(p_3',\ \neg d_p \longrightarrow REP_p^{\ell_p}!,\ p_0')\,.
\end{aligned}
\tag{4.13}
$$

Enfin, l'unique pseudo-transition impliquant le second processus auxiliaire attaché à la station p est

$$
(p_0'',\ REP_q^p? \longrightarrow OR_p^q := 0,\ p_0'')\,. \tag{4.14}
$$

4.7.5. *Réalisation en CSP*

Les programmes exécutés respectivement par le processus principal et par les deux processus auxiliaires sont représentés dans la notation CSP classique à la figure 4.25.

4.7.6. *Vérifications complémentaires*

Le développement méthodique et rigoureux d'un système formel parallèle exige de nombreuses vérifications. Dans le paragraphe précédent, nous en

$$
\begin{array}{l}
*[\ \neg rcs_p \longrightarrow\ [\,] \\
\ \ \Box\ rcs_p \longrightarrow\ (RCS_p, SN_p) := (1, HSN_p + 1); \\
\qquad\qquad *[\Box_{q=1}^{n}\ q \in \mathcal{P}_p \backslash X_p \longrightarrow\ (OR_p^q, X_p) := (1, X_p \cup \{q\})]; \\
\qquad\qquad X_p := \emptyset; \\
\qquad\qquad *[\Box_{q=1}^{n}\ q \in \mathcal{P}_p \backslash X_p \longrightarrow\ REQ_p^q!(SN_p, p);\ X_p := X_p \cup \{q\}]; \\
\qquad\qquad X_p := \emptyset; \\
\qquad\qquad *[\Box_{q=1}^{n}\ q \in \mathcal{P}_p \backslash X_p \wedge \neg OR_p^q \longrightarrow\ X_p := X_p \cup \{q\}]; \\
\qquad\qquad X_p := \emptyset; \\
\qquad\qquad *[rcs_p \longrightarrow\ [\,]]; \\
\qquad\qquad RCS_p := 0; \\
\qquad\qquad *[\Box_{q=1}^{n}\ \ [\ q \in \mathcal{P}_p \backslash X_p \wedge RD_p^q \longrightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad REP_p^q!;\ (RD_p^q, X_p) := (0, X_p \cup \{q\}) \\
\qquad\qquad\qquad\qquad \Box\ q \in \mathcal{P}_p \backslash X_p \wedge \neg RD_p^q \longrightarrow\ X_p := X_p \cup \{q\}\ ]]; \\
\qquad\qquad X_p := \emptyset\ ]\,. \\
*[\ REQ_q^p?(k_p, \ell_p) \longrightarrow\ \ HSN_p := max(HSN_p, k_p); \\
\qquad\qquad\qquad\qquad d_p := (RCS_p \wedge (SN_p, p) \prec (k_p, \ell_p)); \\
\qquad\qquad\qquad\qquad [\ d_p \longrightarrow\ RD_p^{\ell_p} := 1 \\
\qquad\qquad\qquad\qquad \Box\ \neg d_p \longrightarrow\ REP_p^{\ell_p}!\ ]\ ]\,. \\
*[REP_q^p? \longrightarrow\ OR_p^q := 0]\,.
\end{array}
$$

Figure 4.25 *Les trois processus CSP*

avons omis la plus grande part, pour ne pas alourdir l'exposé. Cette omission est en partie réparée dans le présent paragraphe (voir Gribomont 1993 pour plus de détails). Plus précisément, nous allons vérifier que le système $\mathcal{S}_7$, dont la figure 4.24 reprend les transitions, admet l'invariant I_7. Cet invariant se compose de deux parties. L'une d'elles ne concerne que le mécanisme d'horloge et s'écrit

$$I_c =_{def} \forall p \forall q \neq p\, C_{pq}\,,$$

où C_{pq} est la formule

$$SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)\,.$$

Il est commode de procéder en deux temps et de vérifier d'abord l'invariance de I_c.

4.7.6.1. *Exactitude du mécanisme de temps distribué*

L'exactitude de ce mécanisme repose sur les assertions spéciales C_{pq}. Il suffit de prouver que ces assertions forment un invariant, ou encore de prouver la validité des triplets du type $\{\bigwedge_{k \neq l} C_{kl}\}\,\tau\,\{C_{pq}\}$, où τ est n'importe quelle transition. La vérification est immédiate, sauf pour les six transitions suivantes :

$$(p_0 \to p_2)\,,\quad (p_3q'_0 \to p_3q'_0)\,,\quad (p_3q'_0p''_0 \to p_3q'_0p''_0)\,,$$
$$(q_0 \to q_2)\,,\quad (q_3p'_0 \to q_3p'_0)\,,\quad (q_3p'_0q''_0 \to q_3p'_0q''_0)\,.$$

Des preuves de ces triplets (en fait, des triplets plus forts $\{C_{pq}\}\,\tau\,\{C_{pq}\}$) sont résumées ci-dessous.

Transition $(p_0 \to p_2)$.

1. $\begin{pmatrix} \{at^p q_{4560} \Rightarrow SN_q \leq HSN_p\} \\ (RCS_p, SN_p) := (1, HSN_p + 1) \\ \{SN_p = HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)\} \end{pmatrix}$

2. $C_{pq}[at\ p_0] \Rightarrow (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)$

3. $(SN_p = HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)) \Rightarrow C_{pq}[at\ p_2]$

4. $\{C_{pq}\}\,(p_0 \to p_2)\,\{C_{pq}\}$

Transition $(q_0 \to q_2)$.

1. $\begin{pmatrix} \{SN_p \leq HSN_p + 1\} \\ (RCS_q, SN_q) := (1, HSN_q + 1) \\ \{SN_p \leq HSN_p + 1)\} \end{pmatrix}$

2. $C_{pq}[at\ q_0] \Rightarrow SN_p \leq HSN_p + 1$

3. $SN_p \leq HSN_p + 1 \Rightarrow C_{pq}[at\ q_2]$

4. $\{C_{pq}\}\,(q_0 \to q_2)\,\{C_{pq}\}$

Transition $(p_3q'_0 \to p_3q'_0)$.

1. $\begin{pmatrix} \{SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)\} \\ (RD_q^p, HSN_q, X_p) := (1, max(HSN_q, SN_p), X_p \cup \{q\}) \\ \{SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)\} \end{pmatrix}$

2. $C_{pq}[at\ p_3q'_0] \Rightarrow (SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p))$

3. $(SN_p = HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)) \Rightarrow C_{pq}[at\ p_3q'_0]$

4. $\{C_{pq}\}\,(p_3q'_0 \to p_3q'_0)\,\{C_{pq}\}$

Transition $(p_3q'_0p''_0 \to p_3q'_0p''_0)$.

1. $\begin{pmatrix} \{SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)\} \\ (OR_p^q, HSN_q, X_p) := (0, max(HSN_q, SN_p), X_p \cup \{q\}) \\ \{SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)\} \end{pmatrix}$

2. $C_{pq}[at\ p_3q'_0p''_0] \Rightarrow (SN_p \leq HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p))$

3. $(SN_p = HSN_p + 1 \wedge (at^p q_{4560} \Rightarrow SN_q \leq HSN_p)) \Rightarrow C_{pq}[at\ p_3q'_0p''_0]$

4. $\{C_{pq}\}\,(p_3q'_0p''_0 \to p_3q'_0p''_0)\,\{C_{pq}\}$

Transition $(q_3 p'_0 \rightarrow q_3 p'_0)$.

1. $\begin{pmatrix} \{SN_p \leq HSN_p + 1\} \\ (RD_p^q, HSN_p, X_q) := (1, max(HSN_p, SN_q), X_q \cup \{p\}) \\ \{SN_p \leq HSN_p + 1 \wedge SN_q \leq HSN_p\} \end{pmatrix}$

2. $C_{pq}[at\ q_3 p'_0] \Rightarrow SN_p \leq HSN_p + 1$

3. $(SN_p \leq HSN_p + 1 \wedge SN_q \leq HSN_p) \Rightarrow C_{pq}[at\ q_3 p'_0]$

4. $\{C_{pq}\}\,(q_3 p'_0 \rightarrow q_3 p'_0)\,\{C_{pq}\}$

Transition $(q_3 p'_0 q''_0 \rightarrow q_3 p'_0 q''_0)$.

1. $\begin{pmatrix} \{SN_p \leq HSN_p + 1\} \\ (OR_q^p, HSN_p, X_q) := (0, max(HSN_p, SN_q), X_q \cup \{p\}) \\ \{SN_p \leq HSN_p + 1 \wedge SN_q \leq HSN_p\} \end{pmatrix}$

2. $C_{pq}[at\ q_3 p'_0 q''_0] \Rightarrow SN_p \leq HSN_p + 1$

3. $(SN_p \leq HSN_p + 1 \wedge SN_q \leq HSN_p) \Rightarrow C_{pq}[at\ q_3 p'_0 q''_0]$

4. $\{C_{pq}\}\,q_3 p'_0 q''_0 \rightarrow q_3 p'_0 q''_0)\,\{C_{pq}\}$

On peut à présent déduire la validité des triplets de type

$$T_{pq} : \begin{pmatrix} \{SN_q \prec SN_p \vee at^p q_{45}\} \\ (p_0 \rightarrow p_2) \\ \{SN_q \prec SN_p\} \end{pmatrix}.$$

Il sera plus commode de prouver séparément les trois triplets

$\{SN_q < SN_p\}\ v(p_0 \rightarrow p_2)\ \{SN_q < SN_p\}$,

$\{at^p q_{45}\}\ (p_0 \rightarrow p_2)\ \{SN_q < SN_p\}$,

$\{SN_q = SN_p \wedge q < p\}\ (p_0 \rightarrow p_2)\ \{SN_q \leq SN_p \wedge q < p\}$.

Vu l'invariance des assertions de type C_{pq}, la validité de ces triplets est une conséquence immédiate de la validité (évidente) des triplets

$$\begin{pmatrix} \{SN_q < SN_p \wedge SN_p \leq HSN_p + 1\} \\ SN_p := HSN_p + 1 \\ \{SN_q < SN_p\} \end{pmatrix},$$

$$\begin{pmatrix} \{at^p q_{45} \wedge SN_q \leq HSN_p\} \\ SN_p := HSN_p + 1 \\ \{SN_q < SN_p\} \end{pmatrix}$$

et

$$\begin{pmatrix} \{SN_q = SN_p \wedge q < p \wedge SN_p \leq HSN_p + 1\} \\ SN_p := HSN_p + 1 \\ \{SN_q \leq SN_p \wedge q < p\} \end{pmatrix}.$$

4.7.6.2. *Vérification du système formel lui-même*

La deuxième partie de l'invariant global du système est la conjonction, pour tous p, q distincts, des assertions suivantes :

1_p : $[X_p \subset \mathcal{P}_p \wedge (at\ p_{05} \Rightarrow X_p = \emptyset)]$,
2_p : $[at\ p_{06} \equiv \neg RCS_p]$,
3_{pq} : $[(at^q\ p_5 \wedge RCS_q) \Rightarrow SN_p \prec SN_q)]$,
4_{pq} : $[(RD_q^p \Rightarrow OR_p^q) \wedge (\neg at^q\ p_3 \equiv (OR_p^q \Rightarrow RD_q^p))]$,
5_{pq} : $[(at^q\ p_6 \wedge at^p\ q_4) \vee (RD_p^q \equiv (at^p\ q_4 \wedge RCS_p \wedge SN_p \prec SN_q))]$.

Il faut naturellement prouver que cette conjonction forme un invariant. La preuve consiste en la vérification systématique des triplets $\{I\}\tau\{A\}$ pour toute assertion A et pour toute transition τ. Une telle vérification est naturellement fastidieuse, mais en partie automatisable. La validité de la plupart des triplets est évidente, mais il y a quelques cas moins simples; nous en passons quelques-uns en revue. On utilise les abréviations déjà introduites; p et q désignent deux stations distinctes.

Les preuves sont des suites de lignes comportant un numéro d'ordre, une formule assertée et une brève justification. Dans la justification, nous utilisons les abréviations suivantes. "PC" désigne le calcul des propositions, "HL" la logique de Hoare et plus spécifiquement l'axiome de l'affectation, "GA" et "TA" désignent les extensions de cet axiome aux affectations conditionnelles et aux transitions, et "RR" désigne la règle de réduction.

Triplet $\{I\}\,\tau_{02}^q\,\{3_{pq}\}$.

1. $\{at^q p_5\}\,\tau_{02}^q\,\{at^q p_5 \wedge SN_p \prec SN_q\}$ [T_{qp}, HL]
2. $\{\neg at^q p_5\}\,\tau_{02}^q\,\{\neg at^q p_5\}$ [HL]
3. $\{true\}\,\tau_{02}^q\,\{at^q p_5 \Rightarrow SN_p \prec SN_q\}$ [**1**, **2**, HL]
4. $(at^q p_5 \Rightarrow SN_p \prec SN_q) \Rightarrow 3_{pq}[at\ q_2]$ [3_{pq}, RR, PC]
5. $\{I_7\}\,\tau_{02}^q\,\{3_{pq}\}$ [**4**, HL]

Triplet $\{I\}\,\tau_{33a}^{pq}\,\{4_{pq}\}$.

1. $I_7[at\ p_3] \Rightarrow [(RD_q^p \Rightarrow OR_p^q) \wedge (q \in X_p \equiv (OR_p^q \Rightarrow RD_q^p))]$ [4_{pq}]
2. $(I_7[at\ p_3] \wedge q \in \mathcal{P}_p \backslash X_p) \Rightarrow OR_p^q$ [**1**, PC]
3. $\begin{pmatrix}\{I_7[at\ p_3] \wedge q \in \mathcal{P}_p \backslash X_p \wedge RCS_q \wedge SN_q \prec SN_p\} \\ (RD_q^p, X_p) := (1, X_p \cup \{q\}) \\ \{OR_p^q \wedge RD_q^p \wedge q \in X_p \wedge RCS_q \wedge SN_q \prec SN_p\}\end{pmatrix}$ [**2**, HL]
4. $(OR_p^q \wedge RD_q^p \wedge q \in X_p \wedge RCS_q \wedge SN_q \prec SN_p) \Rightarrow 4_{pq}[at\ p_3]$ [4_{pq}, PC, RR]
5. $\begin{pmatrix}\{I_7[at\ p_3] \wedge q \in \mathcal{P}_p \backslash X_p \wedge RCS_q \wedge SN_q \prec SN_p\} \\ (RD_q^p, X_p) := (1, X_p \cup \{q\}) \\ \{4_{pq}[at\ p_3]\}\end{pmatrix}$ [**3**, **4**, HL]
6. $\{I_7\}\,\tau_{33a}^{pq}\,\{4_{pq}\}$ [**5**, GA, TA]

Triplet $\{I\}\,\tau_{33b}^{pq}\,\{4_{pq}\}$.

1. $I_7[at\ p_3] \Rightarrow [(RD_q^p \Rightarrow OR_p^q) \wedge (q \in X_p \equiv (OR_p^q \Rightarrow RD_q^p))]$ [4_{pq}]

2. $(I_7[at\ p_3] \wedge q \in \mathcal{P}_p \backslash X_p) \Rightarrow \neg RD_q^p$ [**1**, PC]

3. $\begin{pmatrix} \{I_7[at\ p_3] \wedge q \in \mathcal{P}_p \backslash X_p \wedge (\neg RCS_q \vee SN_p \prec SN_q)\} \\ (OR_p^q, X_p) := (0, X_p \cup \{q\}) \\ \{\neg OR_p^q \wedge \neg RD_q^p \wedge q \in X_p \wedge (\neg RCS_q \vee SN_p \prec SN_q)\} \end{pmatrix}$ [**2**, HL]

4. $(\neg OR_p^q \wedge \neg RD_q^p \wedge q \in X_p \wedge (\neg RCS_q \vee SN_p \prec SN_q)) \Rightarrow 4_{pq}[at\ p_3]$ [4_{pq}, PC, RR]

5. $\begin{pmatrix} \{I_7[at\ p_3] \wedge q \in \mathcal{P}_p \backslash X_p \wedge (\neg RCS_q \vee SN_p \prec SN_q)\} \\ (OR_p^q, X_p) := (0, X_p \cup \{q\}) \\ \{4_{pq}[at\ p_3]\} \end{pmatrix}$ [**3**, **4**, HL]

6. $\{I_7\}\,\tau_{33b}^{pq}\,\{4_{pq}\}$ [**5**, GA, TA]

Triplet $\{I\}\,\tau_{66a}^{qp}\,\{4_{pq}\}$.

1. $I_7[at\ q_6] \Rightarrow [(RD_q^p \Rightarrow OR_p^q) \wedge (\neg at^q p_3 \equiv (OR_p^q \Rightarrow RD_q^p))]$ [4_{pq}]

2. $(I_7[at\ q_6] \wedge RD_q^p) \Rightarrow \neg at^q p_3$ [**1**, PC]

3. $\begin{pmatrix} \{I_7[at\ q_6] \wedge RD_q^p\} \\ (RD_q^p, OR_p^q, X_q) := (0, 0, X_q \cup \{p\}) \\ \{\neg OR_p^q \wedge \neg RD_q^p \wedge \neg at^q p_3 \wedge p \in X_q\} \end{pmatrix}$ [**2**, HL]

4. $(\neg OR_p^q \wedge \neg RD_q^p \wedge \neg at^q p_3 \wedge p \in X_q) \Rightarrow 4_{pq}[at\ q_6]$ [4_{pq}, PC, RR]

5. $\begin{pmatrix} \{I_7[at\ q_6] \wedge p \in \mathcal{P}_q \backslash X_q \wedge RD_q^p\} \\ (RD_q^p, OR_p^q, X_q) := (0, 0, X_q \cup \{p\}) \\ \{4_{pq}[at\ q_6]\} \end{pmatrix}$ [**3**, **4**, HL]

6. $\{I_7\}\,\tau_{66a}^{qp}\,\{4_{pq}\}$ [**5**, GA, TA]

Triplet $\{I\}\,\tau_{02}^{p}\,\{5_{pq}\}$.

1. $I_7[at\ p_0] \Rightarrow \neg RCS_p$ [2_p, PC, RR]

2. $I_7[at\ p_0] \Rightarrow (RD_p^q \Rightarrow RCS_p)$ [5_{pq}, PC, RR]

3. $I_7[at\ p_0] \Rightarrow \neg RD_p^q$ [**1**, **2**, PC]

4. $\{I_7[at\ p_0]\}\ (RCS_p, SN_p) := (1, HSN_p + 1)\ \{\neg RD_p^q\}$ [**3**, HL]

5. $\{I_7\}\,\tau_{02}^{p}\,\{\neg RD_p^q\}$ [**4**, GA, TA]

6. $\{at^p q_4\}\,\tau_{02}^{p}\,\{SN_q \prec SN_p\}$ [T_{pq}, HL]

7. $\{\neg at^p q_4\}\,\tau_{02}^{p}\,\{\neg at^p q_4\}$ [RR, HL]

8. $\{true\}\,\tau_{02}^{p}\,\{\neg at^p q_4 \vee SN_q \prec SN_p\}$ [T_{pq}, HL]

9. $\{I_7\}\,\tau_{02}^{p}\,\{\neg RD_p^q \wedge \neg(at^p q_4 \wedge SN_p \prec SN_q)\}$ [**5**, **8**, HL]

10. $(\neg RD_p^q \wedge \neg(at^p q_4 \wedge SN_p \prec SN_q)) \Rightarrow 5_{pq}[at\ p_2]$ [5_{pq}, RR, PC]

11. $\{I_7\}\,\tau_{02}^{p}\,\{5_{pq}\}$ [**9**, **10**, HL]

Triplet $\{I\}\,\tau_{02}^{q}\,\{5_{pq}\}$.

1. $I_7[at\ q_0] \Rightarrow \neg RD_p^q$ [5_{pq}, PC, RR]

2. $\{I_7[at\ q_0]\}\ (RCS_q, SN_q) := (1, HSN_q + 1)\ \{\neg RD_p^q\}$ [**1**, HL]

3. $\neg RD_p^q \Rightarrow 5_{pq}[at\ q_2]$ [5_{pq}, RR, PC]

4. $\{I_7[at\ q_0] \wedge rcs_q\}\ (RCS_q, SN_q) := (1, HSN_q + 1)\ \{5_{pq}[at\ q_2]\}$ [**2**, **3**, HL]

5. $\{I_7\}\,\tau_{02}^{q}\,\{5_{pq}\}$ [**4**, GA, TA]

Triplet $\{I\}\,\tau_{33a}^{qp}\,\{5_{pq}\}$.

1. $\begin{pmatrix} \{RCS_p \wedge p \in \mathcal{P}_q \backslash X_q \wedge SN_p \prec SN_q\} \\ (RD_p^q, X_q) := (1, X_q \cup \{p\}) \\ \{RD_p^q \wedge p \in X_q \wedge RCS_p \wedge SN_p \prec SN_q\} \end{pmatrix}$ [HL]

2. $(RD_p^q \wedge p \in X_q \wedge RCS_p \wedge SN_p \prec SN_q) \Rightarrow 5_{pq}[at\ q_3]$ [**1**, PC, RR]

3. $\begin{pmatrix} \{I_7[at\ q_3] \wedge p \in \mathcal{P}_q \backslash X_q \wedge RCS_p \wedge SN_p \prec SN_q\} \\ (RD_p^q, X_q) := (1, X_q \cup \{p\}) \\ \{5_{pq}[at\ q_3]\} \end{pmatrix}$ [**2**, HL]

4. $\{I_7\}\,\tau_{33a}^{qp}\,\{5_{pq}\}$ [**3**, GA, TA]

Triplet $\{I\}\,\tau_{33b}^{qp}\,\{5_{pq}\}$.

1. $I_7[at\ q_3] \Rightarrow [(RD_p^q \Rightarrow OR_q^p) \wedge (p \in X_q \equiv (OR_q^p \Rightarrow RD_p^q))]$ [4_{qp}, RR]

2. $I_7[at\ q_3] \Rightarrow (p \in \mathcal{P}_q \backslash X_q \equiv (OR_q^p \wedge \neg RD_p^q))$ [**2**, PC]

3. $(I_7[at\ q_3] \wedge p \in \mathcal{P}_q \backslash X_q) \Rightarrow \neg RD_p^q$ [**1**, **2**]

4. $\begin{pmatrix} \{\neg RD_p^q \wedge p \in \mathcal{P}_q \backslash X_q \wedge (\neg RCS_p \vee SN_q \prec SN_p)\} \\ (OR_q^p, X_q) := (0, X_q \cup \{p\}) \\ \{\neg RD_p^q \wedge (\neg RCS_p \vee SN_q \prec SN_p)\} \end{pmatrix}$ [**3**, HL]

5. $(\neg RD_p^q \wedge (\neg RCS_p \vee SN_q \prec SN_p)) \Rightarrow 5_{pq}[at\ q_3]$ [5_{pq}, PC, RR]

6. $\begin{pmatrix} \{I_7[at\ q_3] \wedge p \in \mathcal{P}_q \backslash X_q \wedge \neg(RCS_p \vee SN_q \prec SN_p)\} \\ (OR_q^p, X_q) := (0, X_q \cup \{p\}) \\ \{5_{pq}[at\ q_3]\} \end{pmatrix}$ [**4**, **5**, HL]

7. $\{I_7\}\,\tau_{33b}^{qp}\,\{5_{pq}\}$ [**6**, GA, TA]

Triplet $\{I\}\,\tau_{44}^{qp}\,\{5_{pq}\}$.

1. $I_7[at\ q_4] \Rightarrow [(RD_p^q \Rightarrow OR_q^p) \wedge (true \equiv (OR_q^p \Rightarrow RD_p^q))]$ [4_{qp}]

2. $(I_7[at\ q_4] \wedge \neg OR_q^p) \Rightarrow \neg RD_p^q$ [**1**, PC]

3. $\{\neg RD_p^q \wedge p \in \mathcal{P}_q \backslash X_q\}\ X_q := X_q \cup \{p\}\ \{\neg RD_p^q \wedge p \in X_q\}$ [**2**, HL]

4. $(\neg RD_p^q \wedge p \in X_q) \Rightarrow 5_{pq}[at\ q_4]$ [5_{pq}, PC, RR]

5. $\{I_7[at\ q_4] \wedge p \in \mathcal{P}_q \backslash X_q \wedge \neg OR_q^p\}\ X_q := X_q \cup \{p\}\ \{5_{pq}[at\ q_4]\}$ [**3**, **4**, HL]

6. $\{I_7\}\,\tau_{44}^{qp}\,\{5_{pq}\}$ [**5**, GA, TA]

Triplet $\{I\}\ \tau_{56}^{p}\ \{5_{pq}\}$.

1. $I_7[at\ p_5] \Rightarrow (q \in \mathcal{P}_p \backslash X_p \wedge RCS_p)$ [1_p, 2_p, RR, PC]

2. $I_7[at\ p_5] \Rightarrow (RD_p^q \equiv (at^p q_4 \wedge SN_p \prec SN_q))$ [**1**, PC]

3. $I_7[at\ p_5] \Rightarrow ((at^p q_4 \vee \neg RD_p^q) \wedge q \in \mathcal{P}_p \backslash X_p)$ [**1**, **2**, PC]

4. $\begin{pmatrix} \{(at^p q_4 \vee \neg RD_p^q) \wedge p \in \mathcal{P}_q \backslash X_q\} \\ RCS_p := 0 \\ \{(at^p q_4 \vee \neg RD_p^q) \wedge p \in \mathcal{P}_q \backslash X_q \wedge \neg RCS_p\} \end{pmatrix}$ [**3**, HL]

5. $((at^p q_4 \vee \neg RD_p^q) \wedge p \in \mathcal{P}_q \backslash X_q \wedge \neg RCS_p) \Rightarrow 5_{pq}[at\ p_6]$ [5_{pq}, PC, RR]

6. $\{I_7[at\ p_5] \wedge \neg rcs_p\}\ RCS_p := 0\ \{5_{pq}[at\ p_6]\}$ [**4**, **5**, HL]

7. $\{I_7\}\ \tau_{56}^{p}\ \{5_{pq}\}$ [**6**, GA, TA]

4.7.6.3. *Conclusion*

En ce qui concerne la vérification formelle exhaustive des programmes, on observe que la longueur des preuves, déjà importante en programmation séquentielle, devient critique en programmation parallèle. On peut cependant espérer que les progrès de la recherche en démonstration automatique rendront ce problème moins critique dans un proche avenir.

4.8. Calcul distribué d'un point fixe

Nous avons vu que l'approche parallèle permettait d'accélérer des calculs pour lesquels l'approche séquentielle était en principe appropriée. L'exemple du tri systolique (§ 4.3) était de ce type. Un parallélisme plus fondamental survient pour certaines applications qui échappent entièrement au paradigme séquentiel; le problème de l'exclusion mutuelle est un représentant typique de cette catégorie (§ 4.5-4.7). Enfin, nous allons examiner maintenant un problème mixte, dans lequel le parallélisme intervient sous ses deux formes.

4.8.1. *Description du problème*

Soit $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{y})$ une équation dans un espace à n dimensions. Si certaines hypothèses (non précisées ici) sont satisfaites, cette équation de point fixe peut être résolue par approximations successives. On sélectionne une valeur initiale appropriée $\hat{\boldsymbol{y}}$ pour $\boldsymbol{y}$, puis on exécute des affectations du type $y_i := f_i(\boldsymbol{y})$, où $\boldsymbol{y} = (y_1, \ldots, y_n)$ et $\boldsymbol{f}(\boldsymbol{y}) = (f_1(\boldsymbol{y}), \ldots, f_n(\boldsymbol{y}))$, jusqu'à ce que la différence entre $\boldsymbol{y}$ et $\boldsymbol{f}(\boldsymbol{y})$ devienne négligeable. (Cette condition est notée $\boldsymbol{y} \simeq \boldsymbol{f}(\boldsymbol{y})$ et est supposée équivalente à la condition $\forall i\,[y_i \simeq f_i(\boldsymbol{y})]$.) Un système formel parallèle $\mathcal{S}_0$ correspondant à cette technique est représenté à la figure 4.26, avec un invariant I_0.

$\mathcal{P}_0 = \{P_1, \ldots, P_n\}$, où $P_i = \{\ell_n^i\}$, pour tout $i = 1, \ldots, n$,
$\mathcal{M}_0 = \{y_i : i = 1, \ldots, n\}$,
$\mathcal{T}_0 = \{\alpha_i : i = 1, \ldots, n\}$,
où $\alpha_i = (\ell_n^i,\ y_i \not\simeq f_i(\boldsymbol{y}) \longrightarrow y_i := f_i(\boldsymbol{y}),\ \ell_n^i)$;
$I_0 = \mathit{true}$.

Figure 4.26 *Point fixe distribué : système de base*

Remarques. L'unique étiquette du processus P_i est notée ℓ_n^i (plutôt que ℓ^i, par exemple) pour simplifier des notations ultérieures.
Sauf mention explicite du contraire, $\forall i\, A$ abrège $\forall i\, [1 \leq i \leq n \Rightarrow A]$; de même, $\exists i\, A$ abrège $\exists i\, [1 \leq i \leq n \wedge A]$.

On distingue clairement l'un des aspects du parallélisme dans cette application; le calcul du point fixe est accéléré par parallélisation, puisque chaque processus altère une composante du vecteur $\boldsymbol{y}$, en lui appliquant une fonction spécifique. Toutes les composantes de $\boldsymbol{y}$ doivent cependant être accessibles en lecture à chacun des processus.

L'autre aspect du parallélisme apparaît si on se pose le problème de la terminaison, c'est-à-dire de l'arrêt global du calcul. Chaque processus P_i s'arrête lorsque la condition $y_i \simeq f_i(\boldsymbol{y})$ est satisfaite, et redémarre quand, suite à l'intervention des autres processus, cette condition cesse d'être vérifiée. Le problème se pose alors clairement : quand le processus P_i peut-il s'arrêter définitivement ? Ce problème n'est qu'une instance d'un problème général en programmation parallèle, celui de la "terminaison distribuée". Quand plusieurs processus collaborent à une même tâche, comment déterminent-ils que leur contribution au travail commun est achevée ?

La solution présentée dans cette section a été proposée et démontrée par E.W. Dijkstra (1979, voir aussi Best 1979, Chandy et Misra 1986, 1988 et Gribomont 1990). Avant de l'introduire, il est intéressant de rendre les notations plus abstraites. Le problème de la terminaison distribuée et sa solution ne requièrent pas la connaissance des composantes y_i elles-mêmes; il suffit de savoir si l'approximation est de bonne qualité ($y_i \simeq f_i(\boldsymbol{y})$) ou non ($y_i \not\simeq f_i(\boldsymbol{y})$). Ces relations seront représentées respectivement par e_i et $\neg e_i$, où $\boldsymbol{e}$ est donc un vecteur booléen. De même, nous notons A_i l'affectation $y_i := f_i(\boldsymbol{y})$. L'effet de cette instruction sur le vecteur $\boldsymbol{y}$ n'est pas trivial; en particulier, l'exécution de A_i ne garantit pas la postcondition $y_i = f_i(\boldsymbol{y})$. Dans le présent contexte, il n'est pas nécessaire de connaître précisément cet effet; on peut même supposer qu'il est purement aléatoire, ce que l'on modélisera comme suit :

$wlp[A_i; e_j] = \mathit{false}$,
$wlp[A_i; \neg e_j] = \mathit{false}$.

Remarques. L'affectation (déterministe) $y_i := f_i(\boldsymbol{y})$ est donc modélisée par

un programme A_i non déterministe; en particulier, $wlp[A_i; (e_j \vee \neg e_j)]$ est identiquement vrai, tandis que $(wlp[A_i; e_j] \vee wlp[A_i; \neg e_j])$ est identiquement faux. Ce modèle, quoique peu réaliste, est parfaitement acceptable dans le cas présent, puisque nous ne nous intéressons pas aux *performances* de l'algorithme. En fait, l'exigence essentielle est que A_i n'ait d'effet que sur le vecteur $\boldsymbol{e}$, et non sur le vecteur $\boldsymbol{h}$ qui sera introduit plus loin.

4.8.2. *Une solution abstraite*

La solution proposée par Dijkstra (1979) consiste en l'introduction d'un vecteur de nouvelles variables, noté $\boldsymbol{h} = (h_1, \ldots, h_n)$, tel que la variable booléenne h_i soit vraie quand (au moins) une exécution supplémentaire de la transition $(\ell_n^i \rightarrow \ell_n^i)$ est requise. Initialement, tous les h_i sont vrais; une composante h_i peut être fausse seulement si la relation $y_i \simeq f_i(\boldsymbol{y})$ est vraie. La condition de terminaison sera, pour chaque processus, exprimée par la formule $\neg\exists\, j\, h_j$. Chaque fois qu'une instruction A_i est exécutée, toutes les composantes du vecteur $\boldsymbol{h}$ sont mises à vrai; dès que le processus P_i détecte que la condition e_i est vraie, la composante h_i est mise à faux. Ceci donne le système $\mathcal{S}_1$, décrit par l'invariant I_1 (figure 4.27). Dans cette figure, la notation A_B est utilisée pour exprimer que les affectations A et B n'interfèrent pas et peuvent donc être exécutées en même temps ou dans un ordre quelconque. Comme d'habitude, le point-clef de l'algorithme se retrouve dans l'invariant : une composante h_i ne peut être fausse que si la composante correspondante e_i est vraie.

$$
\begin{array}{l}
\mathcal{P}_1 = \{\{\ell_n^i\} : i = 1, \ldots, n\}, \\
\mathcal{M}_1 = \{y_i, h_i : i = 1, \ldots, n\}, \\
\mathcal{T}_1 = \{\beta_i, \gamma_i \,:\, i = 1, \ldots, n\}, \\
\quad \text{où} \;\; \beta_i = \;\; (\ell_n^i,\; \exists\, j\, h_j \wedge e_i \;\longrightarrow\; h_i := \mathit{false},\; \ell_n^i), \\
\qquad\quad \gamma_i = \;\; (\ell_n^i,\; \exists\, j\, h_j \wedge \neg e_i \;\longrightarrow\; A_i _ (\boldsymbol{h} := \boldsymbol{true}),\; \ell_n^i).
\end{array}
$$

$$
I_1 =_{def} \forall\, i\, (h_i \vee e_i).
$$

Figure 4.27 *Une solution abstraite*

4.8.3. *Une solution plus réaliste*

La version $\mathcal{S}_1$ est satisfaisante en principe mais, en pratique, le parallélisme est fort restreint; en particulier, la mise à vrai des composantes du vecteur $\boldsymbol{h}$ doit suivre immédiatement l'exécution d'une instruction A_i, sans interférence des autres processus. Dans ce paragraphe, nous allons relâcher cette exigence de manière progressive. Comme d'habitude, toute modification du système formel est justifiée par une adaptation de l'invariant.

La première modification consiste à remplacer le programme

$$A_i _ (\boldsymbol{h} := \boldsymbol{true})\,, \quad i = 1, \ldots, n\,,$$

considéré comme atomique, par la séquence

$$A_i\,;\ \boldsymbol{h} := \boldsymbol{true}\,, \quad i = 1, \ldots, n\,.$$

On introduit la nouvelle étiquette ℓ_0^i pour désigner le point de contrôle atteint après l'exécution de A_i mais avant celle de $\boldsymbol{h} := \boldsymbol{true}$, par le processus P_i. En ce point, l'invariant initial ne sera plus vérifié, l'effet de A_i sur les composantes de $\boldsymbol{h}$ étant imprévisible. Par contre, on sait que A_i n'est exécuté que si e_i est faux, et donc si h_i est vrai. Cette condition est respectée par A_i, ce qui suggère une adaptation possible de l'invariant. La nouvelle version est représentée avec son invariant à la figure 4.28.

$$\begin{array}{l}
\mathcal{P}_2 = \ \{\{\ell_n^i, \ell_0^i\} : i = 1, \ldots, n\}\,, \\
\mathcal{M}_2 = \{y_i, h_i : i = 1, \ldots, n\}\,, \\
\mathcal{T}_2 = \{\beta_i, \delta_i, \varepsilon_i : i = 1, \ldots, n\}\,, \\
\quad \text{où} \ \ \beta_i \quad (\ell_n^i,\ \exists\, j\, h_j \wedge e_i \ \longrightarrow\ h_i := \mathit{false},\ \ell_n^i)\,, \\
\qquad\ \ \delta_i \quad (\ell_n^i,\ \exists\, j\, h_j \wedge \neg e_i \ \longrightarrow\ A_i\,,\ \ell_0^i)\,, \\
\qquad\ \ \varepsilon_i \quad (\ell_0^i,\ \boldsymbol{h} := \boldsymbol{true}\,,\ \ell_n^i)\,.
\end{array}$$

$$I_2 \quad =_{def} \quad \forall\, i\, (at\ \ell_n^i \wedge (h_i \vee e_i)) \vee \exists\, k\, (at\ \ell_0^k \wedge h_k)\,.$$

Figure 4.28 *Premier raffinement*

La vérification de l'invariance de I_2 est élémentaire. Nous utilisons ici le calcul "SP" pour cette vérification. Les plus fortes postconditions relatives à I_2 et aux transitions de $\mathcal{S}_2$ sont déterminées comme suit :

$$\begin{array}{rl}
slp[I_2;\, \beta_j] \ \equiv & slp[(I_2[at\ \ell_n^j] \wedge e_j);\ h_j := \mathit{false}] \ \wedge\ at\ \ell_n^j\,, \\
\equiv & slp[([(\forall\, i \neq j)\, (at\ \ell_n^i \wedge (h_i \vee e_i)) \wedge (h_j \vee e_j) \wedge e_j] \vee \\
& \qquad [(\exists\, k \neq j)\, (at\ \ell_0^k \wedge h_k) \wedge e_j]);\ h_j := \mathit{false}] \ \wedge\ at\ \ell_n^j\,, \\
\equiv & (slp[((\forall\, i \neq j)\, (at\ \ell_n^i \wedge (h_i \vee e_i)) \wedge e_j);\ h_j := \mathit{false}] \vee \\
& \ slp[((\exists\, k \neq j)\, (at\ \ell_0^k \wedge h_k) \wedge e_j);\ h_j := \mathit{false}]) \ \wedge\ at\ \ell_n^j\,, \\
\equiv & (\forall\, i\, (at\ \ell_n^i \wedge (h_i \vee e_i)) \wedge e_j \wedge \neg h_j) \vee \\
& (at\ \ell_n^j \wedge (\exists\, k \neq j)\, (at\ \ell_0^k \wedge h_k) \wedge e_j \wedge \neg h_j)\,.
\end{array}$$

$$\begin{array}{rl}
slp[I_2;\, \delta_j] \ \equiv & slp[(I_2[at\ \ell_n^j] \wedge \neg e_j);\ A_j] \ \wedge\ at\ \ell_0^j\,, \\
\equiv & slp[([(\forall\, i \neq j)\, (at\ \ell_n^i \wedge (h_i \vee e_i)) \wedge (h_j \vee e_j) \wedge \neg e_j] \vee \\
& \qquad [(\exists\, k \neq j)\, (at\ \ell_0^k \wedge h_k) \wedge \neg e_j]);\ A_j] \ \wedge\ at\ \ell_0^j\,, \\
\equiv & (slp[((\forall\, i \neq j)\, (at\ \ell_n^i \wedge (h_i \vee e_i)) \wedge h_j \wedge \neg e_j);\ A_j] \vee \\
& \ slp[((\exists\, k \neq j)\, (at\ \ell_0^k \wedge h_k) \wedge \neg e_j);\ A_j]) \ \wedge\ at\ \ell_0^j\,, \\
\equiv & ([(\forall\, i \neq j)\, at\ \ell_n^i \wedge h_j] \vee (\exists\, k \neq j)\, (at\ \ell_0^k \wedge h_k)) \wedge e_j \wedge at\ \ell_0^j\,.
\end{array}$$

$$\begin{aligned} slp[I_2;\, \varepsilon_j] &\equiv slp[I_2[at\ \ell_0^j]);\ h := true] \ \wedge\ at\ \ell_n^j\,, \\ &\equiv slp[(false \ \vee\ [h_j \ \vee\ (\exists k \neq j)\,(at\ \ell_0^k \ \wedge\ h_k)]);\ h := true] \ \wedge\ at\ \ell_n^j\,, \\ &\equiv slp[(h_j \ \vee\ (\exists k \neq j)\,(at\ \ell_0^k \ \wedge\ h_k));\ h := true] \ \wedge\ at\ \ell_n^j\,, \\ &\equiv \forall i\, h_i \ \wedge\ at\ \ell_n^j\,. \end{aligned}$$

On voit immédiatement que ces postconditions sont plus fortes que I_2 lui-même, ce qui montre bien que les transitions respectent I_2 (équivalences 3.11).

L'étape suivante consiste à remplacer l'affectation multiple

$$(h_1, \ldots, h_n) := (true, \ldots, true)$$

par une séquence de n affectations simples. Pour chaque processus P_i, de nouvelles étiquettes ℓ_r^i sont introduites ($r = 1, \ldots, n-1$), correspondant aux $(n-1)$ points de contrôle intermédiaires ainsi créés.

L'invariant doit être adapté à nouveau, et un candidat assez naturel est le suivant :

$$\forall i\,(h_i \vee e_i) \ \vee\ \exists k\,(\exists r > 0)\,(at\ \ell_r^k \ \wedge\ h_k \ \wedge\ (\forall i \leq r)\,h_i)\,.$$

Cette assertion n'est pas invariante parce qu'un processus P_i peut falsifier h_i quand e_i est vrai. La version corrigée

$$\begin{aligned} &\forall i\,(h_i \vee e_i) \ \vee\ \exists i\,(at\ \ell_0^i \wedge h_i) \ \vee \\ &(\exists r > 0)\,(at\ \ell_r^k \ \wedge\ h_k \ \wedge\ (\forall i \leq r)\,(h_i \ \vee\ e_i)) \end{aligned}$$

n'est toujours pas satisfaisante; un processus P_j, exécutant l'instruction A_j, pourra la falsifier. Toutefois, cette exécution ne sera possible qu'en l'état ℓ_0^j, et ne sera gênante que si $j > r$. Ceci suggère une troisième adaptation,

$$\begin{aligned} &\forall i\,(h_i \vee e_i) \ \vee\ \exists i\,(at\ \ell_0^i \wedge h_i) \ \vee \\ &(\exists r > 0)\,(at\ \ell_r^k \ \wedge\ h_k \ \wedge\ [(\forall i \leq r)(h_i \vee e_i) \ \vee\ (\exists j > r)\,at\ \ell_0^j])\,. \end{aligned}$$

Cette expression n'est pas encore un invariant, car elle sera falsifiée par l'exécution de la transition $(\ell_0^j,\ h_1 := true,\ \ell_1^j)$, au départ d'un état vérifiant l'assertion

$$(at\ \ell_r^k \ \wedge\ h_k \ \wedge\ j > r \ \wedge\ at\ \ell_0^j)\,.$$

On se trouve ici dans un cas, heureusement très rare en pratique, où l'adaptation de l'invariant semble impossible, alors que le raffinement que cette adaptation est censée justifier semble (et est) licite. En fait, comme nous le démontrerons au chapitre suivant, si l'invariant de départ est bien choisi et si la modification entreprise est licite, l'adaptation de l'invariant est toujours possible. Simplement, elle peut requérir l'introduction de notations supplémentaires, spécifiques au programme traité. La découverte de ces notations peut être assez difficile. Dans le cas présent, l'expression d'un invariant correct requiert les notations suivantes. Pour tout $z \in \{1, \ldots, n\}$, on définit σ_z comme l'ensemble des z-uplets ordonnés d'éléments distincts de l'ensemble $\{1, \ldots, n\}$. Le sous-ensembles formés des multiplets ne contenant pas n est noté σ'_z. De plus, si $\boldsymbol{t} \in \sigma_z$ et si $\boldsymbol{b} \in \sigma'_z$, on définit

$$at\ \ell_b^t =_{def} at\ \ell_{b_1}^{t_1} \wedge \cdots \wedge at\ \ell_{b_z}^{t_z}.$$

On introduit alors quatre assertions spéciales, simplement nommées 1, 2, 3 et 4 :

(4.15)

1. $\forall j\,(h_j \vee e_j)$,
2. $\exists j\,(at\ \ell_0^j \wedge h_j)$,
3. $(\exists z{>}0)(\exists \boldsymbol{t}{\in}\sigma_z)(\exists \boldsymbol{b}{\in}\sigma'_z)$
   $(at\ \ell_b^t \wedge h_{t_z} \wedge (\forall \alpha{<}z)(t_\alpha{>}b_{\alpha+1}) \wedge (\forall j \leq b_1)(h_j \vee e_j))$,
4. $(\exists z{>}0)(\exists \boldsymbol{t}{\in}\sigma_z)(\exists \boldsymbol{b}{\in}\sigma'_z)$
   $(at\ \ell_b^t \wedge h_{t_z} \wedge (\forall \alpha{<}z)(t_\alpha{>}b_{\alpha+1}) \wedge (\exists j{>}b_1)\,at\ \ell_0^j)$.

La disjonction $(1 \vee 2 \vee 3 \vee 4)$ de ces assertions formera un invariant approprié pour le système modifié (voir figure 4.29).

$$\begin{aligned}
\mathcal{P}_3 = {} & \{\{\ell_j^i : j = 0, \ldots, n\} : i = 1, \ldots, n\}, \\
\mathcal{M}_3 = {} & \{y_i, h_i : i = 1, \ldots, n\}, \\
\mathcal{T}_3 = {} & \{(\ell_n^i,\ \exists j\, h_j \wedge e_i \longrightarrow h_i := \mathit{false},\ \ell_n^i), \\
& \quad (\ell_n^i,\ \exists j\, h_j \wedge \neg e_i \longrightarrow A_i,\ \ell_0^i), \\
& \quad (\ell_{j-1}^i,\ h_j := \mathit{true},\ \ell_j^i) : j = 1, \ldots, n : \\
& \quad i = 1, \ldots, n\} \\
I_3 = {} & _{def}\ (1 \vee 2 \vee 3 \vee 4).
\end{aligned}$$

Figure 4.29 *Une version plus réaliste du système*

La preuve complète figure dans Gribomont (1990). Elle est relativement longue mais son principe est tout à fait classique. Il faut montrer que l'exécution de n'importe quelle transition du système, effectuée au départ d'un état vérifiant l'une des quatre assertions du tableau 4.15, conduit à un état vérifiant également une des assertions de ce tableau (pas nécessairement la même). A titre illustratif, nous détaillons ici le cas de la transition $(\ell_{u-1}^p \to \ell_u^p)$, effectuée au départ d'un état satisfaisant l'assertion 3 et, plus précisément, l'assertion

$$at\ \ell_b^t \wedge h_{t_z} \wedge (\forall \alpha{<}z)(t_\alpha{>}b_{\alpha+1}) \wedge (\forall j \leq b_1)(h_j \vee e_j),$$

pour des valeurs données et licites de p, u, z, $\boldsymbol{t}$ et $\boldsymbol{b}$.

On observe d'abord que si $p \notin \boldsymbol{t}$, la transition est sans effet sur les objets évoqués par l'assertion initiale; celle-ci est donc préservée. Un autre cas évident est celui où $p \in \boldsymbol{t}$, par exemple $p = t_v$, mais où $u - 1$ n'a pas la valeur analogue, c'est-à-dire $u - 1 \neq b_v$; dans ce cas, la transition n'est pas exécutable. Le cas non dégénéré survient quand, par exemple, $p = t_v$ et $u - 1 = b_v$; la transition est alors exécutable. On distingue plusieurs sous-cas.

- Si $v > 1$ et $t_{v-1} > b_v + 1$, l'état résultant vérifie l'assertion initiale où b_v est remplacé par $b_v + 1$; en conséquence, l'état résultant vérifie encore

l'assertion 3 et l'invariant.

- Si $v > 1$ et $t_{v-1} = b_v + 1$, l'état résultant vérifie l'assertion initiale où z, $\boldsymbol{t}$ et $\boldsymbol{b}$ sont remplacés respectivement par $v-1$, $\boldsymbol{t}'$ et $\boldsymbol{b}'$, avec $\boldsymbol{t}' = t_1 \ldots t_{v-1}$ et $\boldsymbol{b}' = b_1 \ldots b_{v-1}$. A nouveau, l'assertion 3 et donc l'invariant sont préservés.
- Si $v = 1$, $p = t_1$, $u - 1 = b_1$ et $b_1 + 1 < n$, l'assertion résultante est de nouveau l'assertion initiale où b_v est remplacé par $b_v + 1$.
- Si par contre $v = 1$, $p = t_1$, $u - 1 = b_1$ et $b_1 + 1 = n$, l'état résultant ne vérifie plus l'assertion 3, mais il vérifie l'assertion 1 et donc l'invariant.

Ceci achève la démonstration.

4.8.4. *Autres raffinements*

D'autres modifications sont envisageables, certaines étant licites et d'autres non. Un raffinement licite serait de tester la condition de terminaison générale $\exists\, j h_j$ et la condition spécifique e_i (ou $\neg e_i$) dans des transitions distinctes. Informellement, cela devrait fonctionner, puisque le seul rôle de la garde $\exists\, j h_j$ est de détecter la terminaison. Plus formellement, I_3 reste un invariant de $\mathcal{S}_3$ si le premier terme $\exists\, j\, h_j$ des gardes est supprimé. Le système résultant serait

$$
\begin{aligned}
\mathcal{P}_4 = {} & \{P_i : i = 1, \ldots, n\} \quad \text{où} \quad P_i = \{\ell^i_j : j = 0, \ldots, n+1\}\,, \\
\mathcal{M}_4 = {} & \{y_i, h_i : i = 1, \ldots, n\}\,, \\
\mathcal{T}_4 = {} & \{(\ell^i_n,\ \exists\, j\, h_j \longrightarrow \mathit{skip},\ \ell^i_{n+1})\,, \\
& \quad (\ell^i_{n+1},\ e_i \longrightarrow h_i := \mathit{false},\ \ell^i_n)\,, \\
& \quad (\ell^i_{n+1},\ \neg e_i \longrightarrow A_i,\ \ell^i_0)\,, \\
& \quad (\ell^i_{j-1},\ h_j := \mathit{true},\ \ell^i_j) : j = 1, \ldots, n : \\
& \quad i = 1, \ldots, n\}\,.
\end{aligned}
$$

On pourrait aussi imaginer de scinder les transitions $(\ell^i_{n+1} \to \ell^i_n)$ et $(\ell^i_{n+1} \to \ell^i_0)$ en séquences de deux transitions, la première étant un test et la seconde une affectation non conditionnelle. Le lecteur pourra vérifier que cette scission est licite dans le second cas seulement.

Enfin, il ne semble pas nécessaire de remettre à vrai les composantes h_i dans un ordre fixé et constant. Il est également possible de vérifier que cet ordre est en fait arbitraire, et peut changer en cours d'exécution.

Une fois de plus, un invariant approprié permet de prouver une propriété intéressante et non évidente. Le processus P_k terminera (définitivement) son exécution quand la condition $(at\ \ell^k_n \wedge \neg\exists\, j\, h_j)$ sera vraie, ce qui, vu l'invariant, impliquera que le but global $\forall\, j\, e_j$ est atteint. Il n'y a donc pas de risque d'arrêt "prématuré" d'un processus. En outre, dès que l'un des processus s'arrête, aucune modification de $\boldsymbol{y}$ ne peut plus survenir, et les

autres processus s'arrêteront aussi. Cet exemple illustre bien les qualités de la méthode des invariants, mais aussi les écueils possibles. La découverte ou la construction d'un invariant n'est pas toujours facile et, dans certains cas, peut impliquer l'introduction de nouvelles notations, pour évoquer des objets qui n'apparaissaient pas dans les spécifications du problème.

Chapitre 5
Théorie du point fixe

5.1. Introduction

Ce chapitre introduit d'abord la notion de théorie semi-formelle, dont la théorie du point fixe est un exemple. Les principaux résultats de cette théorie sont alors présentés et appliqués à divers problèmes de l'informatique.

5.1.1. *Théories semi-formelles*

Nous avons examiné au chapitre 9 du volume I les possibilités et les limites des théories du premier ordre. Un certain nombre de résultats limitatifs ont été obtenus; en particulier, on a vu que la plupart des théories ne sont que partiellement décidables. Bien plus, le pouvoir d'expression des langages du premier ordre ne permet pas toujours une modélisation simple des concepts que l'on souhaite étudier. Par exemple, nous avons vu que la notion de bon ordre ne se formalise pas dans le langage de la théorie de l'ordre (qui semblerait pourtant être son cadre naturel). C'est en partie pour remédier à de telles carences que la théorie des ensembles a été introduite au chapitre 2. Cette théorie fournit un cadre général, relativement commode, pour exprimer la grande majorité des concepts mathématiques usuels. On peut appeler *théorie semi-formelle*, ou *théorie ensembliste*, toute spécialisation de la théorie des ensembles dédiée à l'étude d'objets mathématiques particuliers. Une théorie ensembliste peut être vue comme une version partiellement interprétée de la théorie des ensembles.

La logique de Hoare, introduite au chapitre 3, est un bon exemple de théorie ensembliste. Rappelons que le triplet de Hoare

$$\{P\}\ X := E\ \{Q\}$$

est une abréviation de la formule

$$\forall \sigma \forall \rho [(\sigma \models P \wedge (\sigma, \rho) \in R(X := E)) \Rightarrow \rho \models Q] .$$

Rappelons aussi que le symbole $\models$, qui normalement appartient au métalangage, est licite ici, parce qu'il a la signification du symbole $\in$. Les assertions sont en effet assimilées à des ensembles d'états, ce qui permet d'écrire

$$\forall \sigma \forall \rho [(\sigma \in P \wedge (\sigma, \rho) \in R(X := E)) \Rightarrow \rho \in Q] .$$

Le terme "semi-formel" rappelle d'abord que la classe d'objets étudiée par la théorie n'est pas un modèle de la théorie, au sens précis introduit au chapitre I.9. Le terme rappelle en outre que dans les formules d'une théorie ensembliste sont évoqués des objets variés, dont l'appartenance à la classe des ensembles n'est pas toujours évidente, ni même importante.

Dans le cas de la logique de Hoare, les objets étudiés sont les affectations et les programmes. On n'a pas donné à ces objets le statut d'ensemble, mais on les étudie par le biais d'ensembles associés. En effet, la relation $R(S)$ associée au programme S est un ensemble de couples d'états et les assertions sont des ensembles d'états; en outre, un état est une fonction, donc un ensemble de couples.

En fait, une théorie ensembliste est, assez modestement, un *langage* dans lequel on représente commodément les raisonnements non entièrement formels. Les théories de ce type sont classiques en mathématique (analyse, algèbre, topologie, géométrie). Les mathématiciens utilisent ces théories comme outils de communication et de pédagogie. Dans ce contexte, la logique est un langage plus qu'un outil de raisonnement.

5.1.2. *Logique de déduction et logique d'expression*

Classiquement, la logique est l'art — ou la science — du raisonnement. Cette définition reste en bonne partie valable de nos jours, mais a le danger d'occulter un aspect peut-être plus important que le raisonnement. Le premier intérêt de la logique formelle est de permettre une expression précise de faits ou de concepts pour lesquels un haut degré de précision est requis ou souhaité. Avant d'être un outil de déduction, la logique est un langage. Ce rôle de langage existe toujours mais passe au second plan dans les cas — pourtant assez rares — où la logique permet de mécaniser complètement le raisonnement.

Il est intéressant d'observer l'importance relative des deux rôles de la logique dans quelques cas typiques.

- Le domaine du discours fait l'objet d'une théorie du premier ordre décidable. Des exemples classiques sont la théorie de l'ordre linéaire dense, la géométrie classique, la théorie des groupes abéliens et la théorie arithmétique de Presburger (limitée à l'addition). Dans ce contexte, le problème du raisonnement est entièrement résolu, du moins en principe, puisqu'il existe un algorithme (hélas très lent) permettant de décider si

une formule est un théorème. L'expérience montre cependant que ces cas sont rares; bien plus, on sort rapidement du contexte où l'algorithme de décision est valable. Si on s'intéresse par exemple à un groupe abélien particulier, on souhaitera généralement prouver des propriétés propres à ce groupe. Dans ce cas, la théorie des groupes abéliens ne suffira pas, puisqu'elle est incomplète. De même, si on s'intéresse aux sous-groupes des groupes abéliens, la théorie s'avère inopérante puisque la notion de sous-groupe ne se formalise pas dans ce cadre.

- Le domaine du discours fait l'objet d'une théorie du premier ordre simple et naturelle, mais indécidable. La majorité des théories algébriques sont de ce type. Le cas de la théorie des nombres, construite sur les postulats de Peano, est douteux. D'une part, cette théorie semble permettre la déduction des propriétés utiles de l'ensemble des entiers naturels mais, d'autre part, on sait que l'ensemble des conséquences logiques des postulats de Peano est strictement inclus dans l'ensemble des vérités arithmétiques formalisables dans le langage de Peano. Bien plus, les spécialistes des nombres utilisent souvent avec profit des théories beaucoup plus riches pour établir des résultats purement arithmétiques (la théorie des fonctions analytiques à variables complexes, notamment).
- Le domaine du discours fait l'objet d'une théorie du premier ordre, mais le choix des postulats de cette théorie n'est pas clairement dicté par les modèles que l'on souhaite étudier. La théorie axiomatique des ensembles est l'archétype de ce cas. En effet, il n'existe pas de "modèle standard" de la théorie des ensembles, alors que l'ensemble des entiers naturels, muni des opérations classiques, est indubitablement *le* modèle standard de la théorie des nombres.
- Reste enfin le cas où le domaine du discours comporte des objets complexes, qui ne font que partiellement l'objet d'une théorie formelle. Les programmes constituent un bon exemple de tels objets et, comme nous l'avons vu dans les chapitres 3 et 4, les théories semi-formelles sont une solution que la logique propose dans ce cas.[1]

Au fur et à mesure que l'on progresse des cas les plus élémentaires vers les cas les plus élaborés, le rôle de langage formel joué par la logique prend le pas sur son rôle traditionnel d'outil de raisonnement. Le mathématicien considère souvent, d'ailleurs, que le rôle de la logique formelle n'est pas capital dans le processus de raisonnement et de démonstration. L'outil de raisonnement du mathématicien provient plutôt de la logique informelle; c'est ce que Pascal appelait "l'esprit de géométrie".

[1] La *logique dynamique* est une extension entièrement formelle de la logique, conçue pour l'étude des programmes. Cette approche a priori plus rigoureuse est aussi plus difficile, et sort du cadre de cet ouvrage.

5.1.3. *Logique typée*

Actuellement, le mathématicien utilise souvent la théorie naïve des ensembles comme outil de formalisation du raisonnement. Comme nous l'avons vu aux chapitres précédents, la théorie des ensembles pallie une grave lacune des théories du premier ordre, à savoir l'impossibilité de parler des sous-structures d'une structure donnée. L'algébriste spécialiste de la théorie des groupes distinguera une structure G (le groupe qu'il étudie), d'un sous-ensemble $H \subset G$ (le centre de G par exemple); il s'autorisera à quantifier non seulement sur les éléments de G, mais aussi sur les sous-groupes de G, c'est-à-dire sur une classe de sous-ensembles de G. Il pourra même quantifier sur des objets plus complexes, tels les automorphismes de G.

Le cadre habituellement utilisé est celui de la *logique typée*. Dans une logique typée, on utilise simultanément plusieurs domaines d'interprétation ou *types*. Certains d'entre eux seront des domaines "primitifs", c'est-à-dire des ensembles d'individus dans lesquels les variables (individuelles) sont interprétées. On aura aussi des domaines "dérivés", obtenus à partir des domaines primitifs au moyen d'opérations ensemblistes. Si un groupe G est un domaine primitif, la classe de ses sous-groupes formera un domaine dérivé. Cette classe sera introduite comme suit :

$$\mathcal{S} = \{H : H \subset G \wedge \forall x \forall y\,[(x \in H \wedge y \in H) \Rightarrow x.y^{-1} \in H]\}.$$

Les variables d'une théorie typée seront *typées*, c'est-à-dire qu'il leur sera assigné un domaine d'interprétation particulier. A toute constante fonctionnelle à n places seront associés $n+1$ domaines (un pour chacun des arguments et un pour la valeur de la fonction). De même, à toute constante prédicative à n places sont associés n domaines.

Comme nous l'avons vu au chapitre 2, il est généralement possible d'attribuer à chaque élément de chaque domaine un type ensembliste, que nous appellerons *genre* dans ce chapitre, pour éviter tout risque de confusion avec la notion de type d'une variable ou d'une constante. Dans une théorie typée, les éléments de la réunion des domaines primitifs seront de genre 0 (éléments fondamentaux), les sous-ensembles de cette réunion seront de genre 1, et ainsi de suite. On sait que, dans certains cas, cette notion élémentaire de genre doit être remise en question. Nous n'entrerons pas dans ces considérations ici ... et assumerons le risque d'introduire des inconsistances. Ce risque est minime si les objets considérés appartiennent à des types simples et peu nombreux. D'autre part, il est possible de rendre parfaitement rigoureuses ces notions informelles, mais cela sortirait du cadre de cet ouvrage.

D'un point de vue théorique, le principal problème est l'existence du genre des objets évoqués. Considérons par exemple le cas des suites (infinies) de nombres réels. Il est commode de se placer dans une théorie dont les domaines primitifs sont $\mathbf{N}$ et $\mathbf{R}$. Il n'est pas indiqué ici d'identifier certains nombres réels à des naturels, pas plus qu'il n'est pertinent (du moins en général) de confondre

un élément d'une suite et un indice d'un tel élément. Les objets de genre 0 seront donc interprétés dans la *réunion disjointe* de **N** et **R**. Une paire $\{n, r\}$ est de genre 1; un couple $(n, r) = \{\{n, r\}, \{n\}\}$ est de genre 2 et une suite sera donc de genre 3. Cela ne signifie pas que tout objet de genre 3 est une suite de réels; ces suites forment un sous-ensemble strict de l'ensemble des objets de genre 3. Les genres n'ont pas d'utilité technique directe; simplement, un objet dont on ne pourrait pas calculer le genre serait "mal fondé" ... et dangereux à utiliser.

On vérifie facilement que les notions ensemblistes de fonction, de relation, de suite, de produit cartésien, etc. ne posent pas de problème de genre si elles sont appliquées à des objets adéquats. Le genre du résultat de l'application est généralement plus élevé de 2 ou 3 unités que le genre des arguments. (Attention néanmoins à certains cas déjà mentionnés, tel l'intersection d'une classe vide.) Dans la suite, nous admettrons l'existence du genre de tous les objets évoqués, en précisant seulement le type logique.

D'un point de vue pratique, le seul problème consiste à préciser, explicitement ou non, le type des objets introduits. Pour les constantes, le type est explicitement spécifié, mais il peut ne pas l'être pour les variables. Si s est une suite de réels, l'expression $s_x = y$ n'a de sens que si la variable x est de type **N** et si la variable y est de type **R**.

Pour les variables quantifiées, on utilise une notation spéciale. La formule

$$(\forall\, x \in \mathcal{T})\; P(x)$$

exprime que P est vrai de tout objet de type $\mathcal{T}$. Ce type peut être non seulement un domaine (primitif ou dérivé), mais tout sous-ensemble du type spécifié pour le prédicat P. Naturellement, P peut être non seulement un prédicat mais aussi une formule composée, pourvu que les contraintes de type soient respectées. Un moyen simple de donner une sémantique précise à ce type de formule est de la convertir en une formule de la théorie des ensembles, à savoir

$$\forall\, x\, (x \in \mathcal{T} \;\Rightarrow\; P(x))\,.$$

5.2. Théorie du point fixe

5.2.1. *Introduction*

Nous nous sommes déjà servi plusieurs fois de la théorie de l'ordre pour introduire et illustrer divers concepts utiles. Dans la suite de ce chapitre, nous allons étudier et appliquer à l'informatique une extension essentielle de la théorie de l'ordre, à savoir la *théorie du point fixe*. Cette théorie est ainsi nommée parce qu'elle permet de résoudre des équations de la forme

$$X = F(X)\,.$$

Les solutions éventuelles de cette équation sont des éléments sur lesquels la transformation F n'a pas d'effet; on les appelle *points fixes* de l'équation, ou de la transformation F.

Remarque. Une *transformation* d'un ensemble E est une fonction de E dans E.

Dans le cadre de cet ouvrage, nous avons une triple raison d'aborder cette théorie. Tout d'abord, de même que la théorie de l'ordre était un exemple élémentaire, mais utile et éclairant, de la notion de théorie du premier ordre, la théorie du point fixe est un exemple instructif de théorie ensembliste et typée. De plus, les équations de point fixe surviennent fréquemment dans divers domaines des mathématiques pures et appliquées. Enfin, plus concrètement, la théorie du point fixe est indispensable à la compréhension profonde des mécanismes de programmation. Comme nous le verrons, cette théorie permet d'exprimer formellement la sémantique des programmes de toutes espèces.

5.2.2. *Eléments remarquables d'un ensemble ordonné*

Nombre de notions dérivées du concept de relation d'ordre s'introduisent aisément dès lors qu'il est possible d'évoquer non seulement les éléments d'un domaine, mais aussi ses sous-ensembles. Pour introduire ces notions, nous nous plaçons dans une théorie typée dont le domaine primitif est un ensemble ordonné E; d'autres domaines seront introduits en cas de besoin.

Un élément x de E est un *majorant* d'une partie A de E si $y \leq x$, pour tout élément y de A. Ceci se formalise comme suit :

$$(\forall\, x \in E)\,(\forall\, A \subset E)\,[major(x, A) \equiv (\forall\, y \in A)\,(y \leq x)]\,.$$

Cette formule est une définition du prédicat *major*. Elle ne comporte pas de variable libre. La variable x est de type E (type de base) et la variable A est de type $\mathcal{P}(E)$ (on écrit $A \subset E$ plutôt que $A \in \mathcal{P}(E)$). La variable y est de type A; plus exactement, la variable A est interprétée en un sous-ensemble du domaine E; ce sous-ensemble est le type de la variable y.

La notion de *maximum* x d'une partie A de E est classique :

$$(\forall\, x \in E)\,(\forall\, A \subset E)\,[max(x, A) \equiv (x \in A \wedge (\forall\, y \in A)\,(y \leq x))]\,.$$

Les notions de *minorant* et de *minimum* s'introduisent de manière analogue.

Il convient d'observer que la formule ci-dessus n'implique pas que toute partie A de E admet un maximum; un contre-exemple évident est le cas $A = \emptyset$. Par contre, on prouve aisément que le maximum, quand il existe, est unique. Cette proposition peut s'écrire

$$(\forall\, x \in E)\,(\forall\, y \in E)\,(\forall\, A \subset E)\,[(max(x, A) \wedge max(y, A)) \Rightarrow x = y]\,.$$

On prouve cette thèse en la simplifiant. En remplaçant les formes prédicatives $max(x, A)$ et $max(y, A)$ par les formes équivalentes fournies par la définition du prédicat *max*, on peut récrire la thèse comme suit :

$$(\forall\, x \in E)\,(\forall\, y \in E)\,(\forall\, A \subset E)$$
$$[(x \in A \wedge (\forall\, z \in A)\,(z \leq x) \wedge y \in A \wedge (\forall\, z \in A)\,(z \leq y)) \Rightarrow x = y]\,.$$

Le type A étant une restriction (un sous-ensemble) du type E, la thèse se simplifie en

$$(\forall A \subset E)\,(\forall x \in A)\,(\forall y \in A)$$
$$[((\forall z \in A)\,(z \leq x) \;\wedge\; (\forall z \in A)\,(z \leq y)) \;\Rightarrow\; x = y]\,.$$

On sait que $\forall z\, U \;\wedge\; \forall z\, V$ est équivalent à $\forall z\,(U \;\wedge\; V)$; ceci permet d'obtenir

$$(\forall A \subset E)\,(\forall x \in A)\,(\forall y \in A)\,[(\forall z \in A)\,(z \leq x \;\wedge\; z \leq y) \;\Rightarrow\; x = y]\,.$$

On renforce la thèse en affaiblissant l'antécédent de l'implication, c'est-à-dire en remplaçant

$$(\forall z \in A)\,(z \leq x \;\wedge\; z \leq y)$$

par

$$(x \leq x \;\wedge\; x \leq y) \;\wedge\; (y \leq x \;\wedge\; y \leq y)\,,$$

puisque l'on se trouve dans un contexte où $x \in A$ et $y \in A$. La thèse, ainsi renforcée, est une conséquence immédiate de la formule

$$[(x \leq x \;\wedge\; x \leq y) \;\wedge\; (y \leq x \;\wedge\; y \leq y)] \;\Rightarrow\; x = y\,;$$

cette formule est valide, ce qui achève la démonstration.

Remarque. La propriété d'unicité du maximum est intuitivement évidente; on peut donc espérer que sa preuve soit simple. C'est bien le cas, mais il convient d'observer qu'une preuve entièrement formelle serait beaucoup plus longue ... et moins éclairante pour le lecteur. Nous avons utilisé diverses règles d'inférence dérivées sans les prouver, ni même les citer. L'une d'elles est la règle de *généralisation typée*: si $P(x)$ est valide, alors $(\forall x \in E)\,P(x)$ est valide. On prouve de telles règles par le biais de la logique non typée. Dans le cas présent, on a successivement

$$P(x)\,,$$
$$x \in E \;\Rightarrow\; P(x)\,,$$
$$\forall x\,[x \in E \;\Rightarrow\; P(x)]\,,$$
$$(\forall x \in E)\,P(x)\,.$$

La plupart des règles dérivées de logique non typée donnent lieu à des règles analogues en logique typée. La prudence est toutefois de mise: par exemple, on peut inverser l'ordre des quantifications dans la formule $\forall x \forall y\, P(x,y)$, mais il serait absurde de le faire dans la formule $(\forall x \in \mathcal{P}(E))\,(\forall y \in x)\,P(x,y)$.
Le nom de théorie semi-formelle souvent attribué aux théories de la logique typée rappelle aussi que, dans ce contexte, la formalisation des preuves est poussée moins loin que dans le contexte de la logique classique.
Rappelons que, pour l'informaticien, l'avantage principal d'une preuve formelle est la possibilité de vérifier mécaniquement son exactitude. L'exactitude d'une preuve semi-formelle, par contre, n'est établie que par consensus au sein de la communauté mathématique; de même, en dernière analyse, la validité d'une expérience de physique ne sera admise qu'après sa répétition par un nombre suffisant de physiciens dont la qualification est reconnue.

On peut souhaiter réunir les avantages des deux sortes de preuves, et concevoir un système capable de valider (et, par le fait même, de formaliser), des preuves semi-formelles. Ce problème a fait l'objet d'une recherche intensive depuis trente ans et des résultats surprenants ont été obtenus. Néanmoins, les techniques actuellement utilisées en intelligence artificielle sont insuffisantes dans la plupart des cas rencontrés en pratique.

Introduisons enfin un abus d'écriture très commode et (à peu près) sans danger. La formule $max(x, A)$ signifie que x est le maximum de A. On sait que le maximum, s'il existe, est unique. On peut donc noter $max(A)$ ce maximum. Il n'y a guère de problème à utiliser la même notation pour un prédicat à deux arguments et pour une fonction à un argument; par contre, il convient de n'employer la notation $max(A)$ que dans un contexte où l'existence du maximum de A est assurée.
On peut aussi utiliser la notation $major(A)$ pour désigner *l'ensemble* des majorants de A, c'est-à-dire l'ensemble $\{x : x \in E \wedge \forall y (y \in A \Rightarrow y \leq x)\}$. Il n'y a pas de problème d'existence ici; l'ensemble $major(A)$ existe toujours (il peut éventuellement être vide). Rappelons cependant que la notion de majorant dépend de "l'ensemble de référence" E dont A est une partie.

Une notion plus faible que celle de maximum, mais tout aussi importante, est celle de meilleure borne supérieure. Un élément x de E est la *meilleure*, ou la *plus petite borne supérieure* d'une partie A de E s'il est le plus petit des majorants de A. Formellement, on a :

$$sup(x, A) \equiv (major(x, A) \wedge (\forall y \in E)[major(y, A) \Rightarrow x \leq y]).$$

La plus petite borne supérieure de A, ou *supremum* de A, étant le minimum d'un ensemble, est nécessairement unique; elle peut ne pas exister. Si le maximum de A existe, c'est en même temps la plus petite borne supérieure de A; si la plus petite borne supérieure de A existe et appartient à A, c'est le maximum de A. Ce résultat s'exprime par la formule suivante :

$$max(x, A) \equiv (sup(x, A) \wedge x \in A).$$

La démonstration (semi-formelle) est simple et laissée au lecteur.
La notion de meilleure, ou plus grande, borne inférieure (on dit aussi *infimum*), s'introduit comme la notion de plus petite borne supérieure. Quand ces éléments existent, on note $sup(A)$ ou $\bigsqcup A$ la meilleure borne supérieure de A; de même, on note $inf(A)$ ou $\bigsqcap A$ la meilleure borne inférieure de A.
Remarque. On écrit souvent $\bigsqcup_{i \in I} x_i$ au lieu de $\bigsqcup \{x_i : i \in I\}$. On écrit aussi $\bigsqcup_i x_i$, et même $\bigsqcup x_i$, quand cela ne prête pas à confusion.

Les concepts d'*élément maximal* et d'*élément minimal* ont déjà été rencontrés. Si A est un sous-ensemble de E, l'élément $x \in A$ est maximal si $x < y$ n'est vrai pour aucun élément $y \in A$; on a donc

$$maximal(x, A) \equiv (x \in A \wedge (\forall y \in A) \neg (x < y)).$$

On vérifie aisément que si A admet un maximum, ce sera l'unique élément

maximal de A. La réciproque est fausse; si A n'admet pas de maximum, il peut admettre zéro, un ou plusieurs éléments maximaux.
La notion d'*élément minimal* se définit de manière analogue.

Nous avons déjà utilisé aussi la notion de *chaîne*; une chaîne de E est un sous-ensemble totalement ordonné de E. Par opposition, une *anti-chaîne* est une partie totalement disconnectée de E; un sous-ensemble A est une anti-chaîne si $x, y \in A \Rightarrow \neg(x < y \vee y < x)$. L'ensemble $\mathcal{C}_E$ des chaînes de E est

$$\mathcal{C}_E = \{C \subset E : (\forall x, y \in C)\,(x \leq y \vee y \leq x)\}\,.$$

Un sous-ensemble A de E est *dirigé* si toute partie finie non vide B de A admet un majorant dans A. Les chaînes sont des ensembles dirigés particuliers; on vérifie immédiatement qu'un sous-ensemble A de E est une chaîne si toute partie finie non vide B de A admet un majorant *et un minorant* dans B (et pas seulement dans A); ces éléments sont naturellement le maximum et le minimum de B.
Remarque. Le vide, vu comme sous-ensemble de E, est une chaîne et un ensemble dirigé. (Certains auteurs excluent explicitement ce cas.)

Rappelons aussi que l'ensemble E est *bien fondé* si toute partie non vide de E admet un élément minimal; il est *bien ordonné* si toute partie non vide de E admet un minimum. Tout ensemble ordonné fini est bien fondé, et un ensemble bien ordonné est un ensemble bien fondé et totalement ordonné. Nous avons vu au chapitre 2 que l'intérêt majeur de ces notions est de permettre l'induction, et nous avons évoqué le théorème de base, que nous rappelons maintenant avec sa démonstration.

Théorème 5.1. Un ensemble ordonné E est bien fondé si et seulement si toutes ses suites strictement décroissantes sont finies.
Démonstration. La condition est nécessaire. Soit $(a_n : n \in \mathbf{N})$ une suite infinie décroissante dans E, donc telle que $(a_{n+1} \leq a_n)$ pour tout n. Soit a un élément minimal (en fait, nécessairement le minimum) de l'ensemble sous-jacent $\{a_n : n \in \mathbf{N}\}$; il existe (au moins) un indice r tel que $a = a_r$, ce qui implique $a_n = a$ pour tout $n \geq r$. La suite (a_n) est donc stationnaire[2] et n'est donc pas strictement décroissante.
La condition est aussi suffisante. Soit A un sous-ensemble infini de E. On peut associer à tout $x \in A$ l'ensemble $A_x = \{y \in A : y < x\}$. Si A n'admet pas d'élément minimal, aucun A_x n'est vide, et on peut définir par récurrence une suite infinie strictement décroissante dans A en choisissant un élément arbitraire $x_0 \in A$ et, pour tout n, un élément arbitraire $x_{n+1} \in A_{x_n}$.
Remarques. La deuxième partie de la démonstration utilise l'axiome du choix. On voit clairement pourquoi les définitions inductives de fonctions requièrent

[2]La suite $(a_n : n \in \mathbf{N})$ est *stationnaire* s'il existe $k \in \mathbf{N}$ tel que $a_{k+n} = a_k$, pour tout n.

que le domaine soit bien fondé. L'idée est de définir la valeur $f(x)$ en fonction de l'ensemble $\{f(y) : y \in A_x\}$; ce n'est licite que si toute "régression" est finie. Un élément x est minimal si $A_x = \emptyset$; dans ce cas, $f(x)$ sera défini directement (cas de base) sans recours aux valeurs de f pour d'autres éléments de E.

Pour illustrer les notions introduites dans ce paragraphe, considérons la relation "divise", définie comme suit :

$$(\forall a, b \in \mathbf{Z})\,[a \mid b \equiv (\exists c \in \mathbf{Z})\,(a.c = b)]\,.$$

Remarque. La sémantique ensembliste de la formule $(\exists x \in \mathcal{T})\,P(x)$ est $\exists x\,(x \in \mathcal{T} \wedge P(x))$.

La relation "divise" dans $\mathbf{Z}$ n'est pas une relation d'ordre, parce qu'elle n'est pas antisymétrique; on a en effet $3 \mid -3$ et $-3 \mid 3$. Dans $\mathbf{N}$ par contre, cette relation est bien réflexive, antisymétrique et transitive; le minimum est 1 et le maximum est 0, car 1 divise tout nombre, et tout nombre divise 0. Le sous-ensemble $\mathbf{N} \setminus \{0, 1\}$ n'admet pas de minimum, mais les éléments minimaux sont exactement les nombres premiers. Il n'y a pas d'élément maximal. Soit A un sous-ensemble non vide de $\mathbf{N} \setminus \{0\}$. La meilleure borne inférieure de A existe toujours : c'est le plus grand commun diviseur des éléments de A. La meilleure borne supérieure de A existe si et seulement si A est fini; c'est alors le plus petit commun multiple des éléments de A. Observons aussi que la relation "divise" est bien fondée dans $\mathbf{N}$. Cet ensemble est dirigé, de même que tout sous-ensemble comportant 0, ou tout sous-ensemble comportant un commun multiple de toute paire de ses éléments. Ces ensembles dirigés ne sont généralement pas des chaînes. L'ensemble des puissances de 2 (ou d'un nombre quelconque) est une chaîne. L'ensemble des nombres premiers est une anti-chaîne.

5.2.3. *La relation d'inclusion*

Soit E un ensemble non vide quelconque. L'ensemble $\mathcal{P}(E)$ est ordonné pour la relation d'inclusion. Toute partie A de cet ensemble, c'est-à-dire toute famille de sous-ensembles de E, admet une meilleure borne supérieure $\bigsqcup A = \bigcup A$ et une meilleure borne inférieure $\bigsqcap A = \bigcap A$. (Ces bornes n'appartiennent pas nécessairement à A.)

La relation d'inclusion est le prototype des relations d'ordre. Bien plus, toute relation d'ordre peut être assimilée à la relation d'inclusion sur un ensemble particulier. Soit $(E, \leq)$ un ensemble ordonné quelconque. Pour tout élément x de E on définit l'*idéal* $\downarrow x$ et le *filtre* $\uparrow x$ associés à x.

$$\downarrow x =_{def} \{y : y \in E \wedge y \leq x\}\,, \quad \uparrow x =_{def} \{y : y \in E \wedge x \leq y\}\,.$$

On note $\mathcal{S}(E)$ l'ensemble des idéaux associés aux éléments de E.[3]

[3] La notation $\mathcal{S}(E, \leq)$ serait plus correcte.

On observe immédiatement que les ensembles ordonnés $(E, \leq)$ et $(\mathcal{S}(E), \subset)$ sont isomorphes. On a bien

$$(\forall\, x, y \in E)\,(x \leq y \;\equiv\; \downarrow x \subset \downarrow y)\,;$$

on a aussi

$$(\forall\, a, b \in \mathcal{S}(E))\,(a \subset b \;\equiv\; max(a) \leq max(b))\,.$$

La notion d'idéal associé à un élément se généralise. D'une part, on appelle *idéal* de $(E, \leq)$ (sans référence à un élément particulier) tout sous-ensemble de E contenant ses minorants. Un idéal contenant x contient aussi tous les éléments plus petits que x. (Par contre, un idéal peut ne plus admettre de maximum ou d'éléments maximaux.) Quand E est fini, un idéal I de E est caractérisé par l'ensemble A de ses éléments maximaux. On écrit alors $I = \downarrow A$. L'ensemble A est une anti-chaîne. L'idéal $\downarrow\{x\}$ s'identifie avec l'idéal $\downarrow x$ associé à x dans l'ensemble E. La notion duale de *filtre* se généralise de même.

Quand l'ensemble E est totalement ordonné, un idéal est appelé *segment initial*. On a vu au chapitre 2 que les ordinaux se confondent avec les segments initiaux propres de la classe des ordinaux; si α est un ordinal, les ordinaux inférieurs à α se confondent avec les segments initiaux de α.

Remarques. Les notions d'idéal et de filtre sont génériques et s'emploient aussi à propos de structures plus riches que celle d'ensemble ordonné (algèbre de Boole, topologie, ...). On reviendra plus loin sur ce point.
Le fait que tout ensemble ordonné soit isomorphe à un ensemble (en fait, une famille d'ensembles) ordonné par inclusion permet d'étudier les relations d'ordre au départ de la seule relation d'inclusion. Même si on étudie une relation d'ordre particulière, au contenu intuitif bien distinct de l'inclusion, il est souvent utile de s'interroger sur la relation d'inclusion associée.

5.2.4. *Points fixes des fonctions croissantes*

Soient E et F deux ensembles ordonnés. Une fonction f de domaine E et de codomaine inclus dans F est *croissante*, ou *monotone*, si elle vérifie

$$(\forall\, x, y \in E)\,(x \leq y \;\Rightarrow\; f(x) \leq f(y))\,.$$

Remarque. Dans cette formule, la première occurrence du symbole $\leq$ désigne la relation d'ordre dans E, tandis que la seconde occurrence désigne la relation d'ordre dans F. L'emploi d'un seul symbole pour représenter deux relations en général distinctes n'introduit habituellement pas de confusion. Dans le cas contraire, on peut utiliser des notations telles que $\leq_E$ et $\leq_F$.

On s'intéresse plus particulièrement aux fonctions croissantes d'un ensemble ordonné sur lui-même; en effet, pour de telles fonctions, il semble possible de calculer un point fixe par approximations successives. Supposons donc que f soit une fonction croissante de domaine E et de codomaine inclus dans E; soit aussi un élément x_0 de E tel que $x_0 \leq f(x_0)$ (il est clair que si aucun élément

de ce type n'existe, la fonction f n'admet pas de point fixe: on a $f(x) < x$, pour tout x). On considère alors $C = \{x_0, x_1 = f(x_0), \ldots, x_{n+1} = f(x_n), \ldots\}$. L'ensemble C est visiblement une chaîne; si cette chaîne est finie, son maximum existe et est un point fixe de f. En effet, si la suite (x_n) est stationnaire, on doit avoir, à partir d'un certain n, $x_n = x_{n+k}$, pour tout $k \in \mathbf{N}$. Malheureusement, la chaîne C peut être infinie; dans ce cas, rien ne garantit que sa "limite", c'est-à-dire sa meilleure borne supérieure, existe et est un point fixe de f.

Considérons l'exemple suivant. La fonction successeur dans $\mathbf{N}$ est croissante mais n'admet pas de point fixe. La chaîne C est ici l'ensemble des naturels supérieurs à un certain naturel n_0; cette chaîne n'admet pas de meilleure borne supérieure. Si par contre on introduit un "élément à l'infini", supérieur à tous les naturels et noté ∞, et si on pose $succ(\infty) = \infty$, alors la fonction *succ* reste croissante et la chaîne C admet la meilleure borne supérieure ∞, qui est un point fixe (le seul) de la fonction *succ*. Cet exemple peut paraître trivial: on a artificiellement "créé" un point fixe pour une fonction qui n'en avait pas. En fait, cet exemple primitif suggère que l'existence de la meilleure borne supérieure conditionne l'existence du point fixe.

Un exemple plus instructif est le suivant. On considère l'ensemble

$$E = \{x \in \mathbf{Q} : 1 \leq x \leq 3/2\}.$$

On munit E d'une relation d'ordre quelque peu inhabituelle, en posant

$$x \prec y \;=_{def}\; |x^2 - 2| > |y^2 - 2|.$$

On considère alors la fonction f, de domaine E, qui à x associe $x/2 + 1/x$. On peut vérifier que le codomaine de f est inclus dans E; en outre, la fonction f est croissante pour la relation d'ordre $\preceq$. Si on pose $x_0 = 1$, on détermine une chaîne C infinie, n'admettant pas de meilleure borne supérieure. Si par contre E est défini comme un intervalle de réels et non comme un intervalle de rationnels, la meilleure borne supérieure existe; c'est le nombre $\sqrt{2}$, qui est l'unique point fixe de la fonction f dans l'intervalle réel $[1, 3/2]$. Cet exemple est plus intéressant que le précédent, car il fournit un moyen (classique) de calculer $\sqrt{2}$ par approximations rationnelles successives.

Evoquons encore un exemple bien connu des utilisateurs de calculettes. Si au départ de l'affichage 0, on appuie répétitivement sur la touche COS (la machine étant en mode radian), on génère la séquence

$$(0,\ 1,\ 0.540300230,\ 0.85755321,\ 0.65428979,\ 0.79348035,\ \ldots).$$

Cette suite oscillante converge assez rapidement vers 0.73908513..., qui est solution de l'équation $cos(x) = x$. Comme dans le cas précédent, la relation d'ordre sous-jacente n'est pas l'ordre numérique habituel.

Le théorème suivant formalise cette technique de calcul du point fixe, mais fait apparaître une difficulté inattendue.

Théorème 5.2. Si E est un ensemble ordonné et si f est une fonction croissante de domaine E et de codomaine inclus dans E, s'il existe un élément x_0

tel que $x_0 \leq f(x_0)$ et si toute chaîne de E admet une meilleure borne supérieure dans E, alors la fonction f admet un point fixe.

Démonstration. On utilise le schéma de von Neumann (§ 2.5.5) pour définir une fonction $f\uparrow$ de domaine Ω et de codomaine inclus dans E. On pose

$$f\uparrow(0) = x_0\,,$$
$$f\uparrow(r') = f(f\uparrow(r))\,, \text{pour tout ordinal } r\,,$$
$$f\uparrow(\ell) = \bigsqcup\{f\uparrow(\alpha) : \alpha \in \ell\}\,, \text{pour tout ordinal limite } \ell\,.$$

La définition est licite, parce que l'ensemble $\{f\uparrow(\alpha) : \alpha \in \ell\}$ est une chaîne, quel que soit l'ordinal ℓ. La fonction $f\uparrow$ ne peut être injective puisque son domaine est une classe propre et son codomaine un ensemble. L'argumentation habituelle permet d'affirmer l'existence d'un plus petit ordinal α tel que $f\uparrow(\alpha') = f\uparrow(\alpha)$; par définition de $f\uparrow$, $f\uparrow(\alpha)$ est un point fixe de f.

Le problème est que la suite des approximations successives $(x_0, x_1, \ldots, x_n, \ldots)$ ne converge pas nécessairement vers un point fixe; le point fixe peut n'être atteint que pour un ordinal plus grand que ω. En outre, une fonction peut admettre plusieurs points fixes; le théorème ne caractérise pas le point fixe atteint par induction ordinale. Pour remédier à ces problèmes, il faut imposer certaines restrictions à E et f. On introduit d'abord une définition. L'ensemble ordonné E est *inductif* s'il admet un minimum (noté $\bot$) et si toute chaîne A de E admet une meilleure borne supérieure dans E (notée $\bigsqcup A$).[4]

Remarque. Beaucoup d'auteurs utilisent "complet" ou "continu" au lieu de "inductif".[5] Nous préférons éviter l'attribution d'une nouvelle signification à des mots qui en ont déjà plusieurs; d'autre part, il semble naturel de réserver le mot "complet" aux ensembles ordonnés dont *tous* les sous-ensembles admettent une meilleure borne supérieure.[6] L'incomplétude d'un ensemble ordonné inductif est cependant relative, comme le montre le résultat suivant.

Théorème 5.3. Dans un ensemble ordonné inductif, tout ensemble dirigé admet une meilleure borne supérieure.

Démonstration. On se limite ici au cas dénombrable. Soit un sous-ensemble dirigé $D = \{x_n : n \in \mathbf{N}\}$ de l'ensemble ordonné inductif E. On définit une suite (y_n) d'éléments de D comme suit: $y_0 = x_0$ et $y_{n+1} = x_k$, où k est le plus petit indice tel que x_k soit un majorant de la paire $\{x_{n+1}, y_n\}$. On vérifie par récurrence que la suite (y_n) est croissante et que, pour tout n, y_n est un majorant de $\{x_0, \ldots, x_n\}$. La meilleure borne supérieure de la chaîne $\{y_n : n \in \mathbf{N}\}$ est la meilleure borne supérieure de D.

---

[4] La première condition est redondante si on admet la chaîne vide: $\bigsqcup\emptyset = \bot$.

[5] Les mêmes usages existent en anglais, et un ensemble ordonné inductif est un CPO ou un IPO (*Continuous, Complete, Inductive Partial Order*).

[6] Dans de tels ensembles, tout sous-ensemble admet aussi une meilleure borne inférieure; en effet, la meilleure borne inférieure d'un sous-ensemble A de E est la meilleure borne supérieure de l'ensemble des minorants de A dans E.

Remarque. Ce théorème explique la coexistence de deux tendances pour développer la théorie du point fixe, en privilégiant la notion de chaîne ou celle, plus générale, d'ensemble dirigé.

Le caractère inductif d'un ensemble ordonné rend triviale, au moins en théorie, la construction d'un point fixe d'une fonction croissante. Si on pose $x_0 = \bot$, il existe un plus petit ordinal α tel que $f\uparrow(\alpha)$ soit un point fixe. Ce point fixe est le plus petit point fixe; en effet, si d est un point fixe, on a $\bot \leq d$ et, par induction ordinale, $f\uparrow(r) \leq d$ pour tout ordinal r. On a donc le résultat suivant.

Corollaire 5.4. L'ensemble des points fixes d'une fonction croissante d'un ensemble ordonné inductif sur lui-même est non vide et admet un minimum.
Remarque. Quand $x_0 = \bot$, l'approximation de rang r du plus petit point fixe de la fonction croissante f est souvent écrite $f\uparrow r$ (sans parenthèses).

Il est parfois possible d'éviter le processus d'approximations successives pour obtenir le plus petit point fixe d'une fonction. On a le théorème suivant.

Théorème 5.5. Soit f une fonction croissante de l'ensemble ordonné E sur lui-même et soit $A = \{x \in E : f(x) \leq x\}$. Si la meilleure borne inférieure $\bigsqcap A$ de A existe, alors c'est le minimum de A et le plus petit point fixe de f.
Démonstration. Soit $m = \bigsqcap A$; on a $m \leq x$, pour tout $x \in A$, et aussi $f(m) \leq f(x)$, d'où, par définition de A, $f(m) \leq x$ pour tout $x \in A$. L'élément $f(m)$ est donc un minorant de A et, puisque m est le plus grand des minorants, on a $f(m) \leq m$. L'infimum m est donc un minimum. On a aussi $f(f(m)) \leq f(m)$ et $f(m) \in A$, ce qui implique $m \leq f(m)$ et $m = f(m)$. Enfin, tout point fixe d de f est un élément de A, donc m est bien le plus petit point fixe de f.
Remarque. Ce théorème n'est pas d'application aisée, car il exige l'existence de la meilleure borne inférieure de l'ensemble A, souvent difficile à vérifier en pratique.

5.2.5. *Points fixes des fonctions continues*

On peut obtenir des résultats plus commodes en matière de points fixes en renforçant quelque peu l'hypothèse de croissance de la fonction f. On introduit d'abord une définition.
Une fonction f dont le domaine est un ensemble ordonné inductif E et dont le codomaine est inclus dans un ensemble ordonné inductif F est *continue* si pour toute chaîne C de E on a $\bigsqcup f(C) = f(\bigsqcup C)$, où $f(C)$ désigne l'ensemble $\{f(x) : x \in C\}$.
Remarques. Si C est une chaîne dans E, alors $f(C)$ est une chaîne dans F.
L'égalité $\bigsqcup f(C) = f(\bigsqcup C)$ reste vraie pour tout ensemble dirigé C.

Le théorème suivant établit la correspondance entre les notions de croissance et de continuité.

Théorème 5.6. Toute fonction continue est croissante.
Démonstration. Soit f une fonction continue de l'ensemble ordonné inductif E dans l'ensemble ordonné inductif F et soient deux éléments x, y de E tels que $x \leq y$. On considère la chaîne finie $\{x, y\}$. L'élément $f(x) \in F$ appartient à la chaîne $\{f(x), f(y)\}$ et est donc inférieur au supremum de cette chaîne qui est $\bigsqcup\{f(x), f(y)\} = f(\bigsqcup\{x, y\}) = f(y)$.
Remarque. Si f est une fonction croissante mais non continue, on ne peut plus écrire $\bigsqcup f(C) = f(\bigsqcup C)$, mais on a toujours $\bigsqcup f(C) \leq f(\bigsqcup C)$. En effet, $f(\bigsqcup C)$ est un majorant de l'ensemble $f(C)$, en raison de la croissance de f, tandis que, par définition, $\bigsqcup f(C)$ est le plus petit majorant de ce même ensemble.

Le plus petit point fixe d'une fonction continue admet une caractérisation très simple.

Théorème 5.7. Le plus petit point fixe d'une fonction continue f d'un ensemble ordonné inductif sur lui-même est $m = f{\uparrow}\omega = \bigsqcup\{f{\uparrow}n : n \in \mathbf{N}\}$.
Démonstration. On a en effet $f(\bigsqcup\{f{\uparrow}n : n \in \mathbf{N}\}) = \bigsqcup\{f(f{\uparrow}n) : n \in \mathbf{N}\} = \bigsqcup\{f{\uparrow}n : n \in \mathbf{N}_0\} = \bigsqcup\{f{\uparrow}n : n \in \mathbf{N}\}$, ce qui prouve que m est bien un point fixe. D'autre part, f étant croissante, on vérifie par induction (ordinaire) que tout point fixe d de f vérifie $f{\uparrow}n \leq d$; d est donc un majorant de la chaîne $\{f{\uparrow}n : n \in \mathbf{N}\}$. Comme m est le plus petit des majorants de cette chaîne, on a $m \leq d$.
Signalons aussi que la composition de fonctions croissantes (continues) donne une fonction croissante (continue).

Nous utiliserons plus loin la continuité dans le cadre d'espaces fonctionnels, composés de fonctions dont le domaine et le codomaine sont ordonnés. Si D est un ensemble et si E est un ensemble ordonné, l'ensemble $(D \to E)$ des fonctions de domaine D et de codomaine inclus dans E est naturellement muni de la relation d'ordre *canonique induite* par $(E, \leq)$, définie comme suit :

$$f \leq g \;=_{def}\; (\forall\, x \in D)\,(f(x) \leq g(x))\,.$$

Remarque. Le même symbole désigne les relations d'ordre dans E et $(D \to E)$.

Si $\{f_i : i \in I\}$ est un sous-ensemble de $(D \to E)$, on pose

$$(\bigsqcap_i f_i)(x) \;=_{def}\; \bigsqcap\{f_i(x) : i \in I\}\,,$$
$$(\bigsqcup_i f_i)(x) \;=_{def}\; \bigsqcup\{f_i(x) : i \in I\}\,,$$

pour tout $x \in D$. Cette notation (classique) est un peu dangereuse, car les fonctions $\bigsqcap_i f_i$ et $\bigsqcup_i f_i$ ainsi définies ne seront les meilleures bornes de $\{f_i : i \in I\}$ que si elles appartiennent à l'ensemble de référence $(D \to E)$, c'est-à-dire si elles sont définies en tout point de D. A ce propos, on a les théorèmes suivants.

Théorème 5.8. Tout sous-ensemble de $(D \to E)$ admet une meilleure borne inférieure (supérieure) si et seulement si tout sous-ensemble de E admet une meilleure borne inférieure (supérieure).

Démonstration. La condition est nécessaire. Soit A un sous-ensemble de E. Pour tout $a \in A$ on note f_a la fonction (constante) qui à tout élément de D associe la valeur a. La fonction $\bigsqcap_{a\in A} f_a$ existe par hypothèse et est constante; l'unique valeur de son codomaine est $\bigsqcap A$.
La condition est suffisante. Soit $\{f_i : i \in I\}$ un sous-ensemble de $(D \to E)$. Si la valeur $\bigsqcap\{f_i(x) : i \in I\}$ existe pour tout $x \in D$, la fonction $\bigsqcap_i f_i$ a pour domaine D et appartient bien à l'ensemble $(D \to E)$.

Théorème 5.9. Soient D et E deux ensembles ordonnés et soit $\{f_i : i \in I\}$ un sous-ensemble de $(D \to E)$ composé de fonctions croissantes. Si elles existent, les meilleures bornes de cet ensemble sont des fonctions croissantes.
Démonstration. Pour la meilleure borne inférieure, par exemple, on a successivement, pour $x, y \in D$,

$$
\begin{array}{l}
x \leq y\,, \\
f_i(x) \leq f_i(y)\,, \text{pour tout } i \in I\,, \\
\bigsqcap\{f_i(x) : i \in I\} \leq f_i(y)\,, \text{pour tout } i \in I\,, \\
\bigsqcap\{f_i(x) : i \in I\} \leq \bigsqcap\{f_i(y) : i \in I\}\,, \\
(\bigsqcap_{i\in I} f_i)(x) \leq (\bigsqcap_{i\in I} f_i)(y)\,.
\end{array}
$$

On a un résultat plus fort pour les fonctions continues, très utile pour les applications. Si D et E sont des ensembles ordonnés inductifs, on note $[D \to E]$ l'ensemble des fonctions continues de D dans E, muni de l'ordre canonique induit par E.

Théorème 5.10. L'ensemble ordonné $[D \to E]$ est inductif.
Démonstration. On observe d'abord que le minimum de $[D \to E]$ est la fonction constante qui à tout élément de D associe le minimum de E. On montre ensuite que toute chaîne $\{f_i : i \in I\}$ de $[D \to E]$ admet une meilleure borne supérieure, qui est naturellement $\bigsqcup_{i\in I} f_i$. Cette fonction est définie en tout point $d \in D$; sa valeur est $\bigsqcup_i f_i(d)$. Il reste à prouver que cette fonction appartient à $[D \to E]$, c'est-à-dire qu'elle est continue.

Soit $\{x_j : j \in J\}$ une chaîne de D. Il faut prouver

$$(\bigsqcup_i f_i)(\bigsqcup_j x_j) \;=\; \bigsqcup_j (\bigsqcup_i f_i)(x_j)\,.$$

On a d'abord le développement suivant :

$$
\begin{array}{lcll}
(\bigsqcup_i f_i)(\bigsqcup_j x_j) & = & \bigsqcup_i f_i(\bigsqcup_j x_j) & [\text{définition de } \bigsqcup_i f_i] \\
& = & \bigsqcup_i (\bigsqcup_j f_i(x_j)) & [\text{continuité de } f_i]\,.
\end{array}
$$

On observe ensuite que,

$$
\begin{array}{l}
f_i(x_j) \leq \bigsqcup_j (\bigsqcup_i f_i(x_j))\,, \text{ pour tout } i \text{ et pour tout } j\,, \\
\bigsqcup_j f_i(x_j) \leq \bigsqcup_j (\bigsqcup_i f_i(x_j))\,, \text{ pour tout } i\,, \\
\bigsqcup_i (\bigsqcup_j f_i(x_j)) \leq \bigsqcup_j (\bigsqcup_i f_i(x_j))\,,
\end{array}
$$

d'où on déduit

$$(\bigsqcup_i f_i)(\bigsqcup_j x_j) \leq \bigsqcup_j (\bigsqcup_i f_i(x_j))\,.$$

L'inégalité inverse a lieu dès que la fonction $\bigsqcup_i f_i$ est croissante; c'est bien le cas puisque les fonctions f_i sont continues, donc croissantes (théorème 5.6) et que le supremum d'un ensemble de fonctions croissantes est une fonction croissante (théorème 5.9).

On peut considérer les opérateurs $\uparrow n$, pour tout $n \in \mathbf{N}$, et l'opérateur $\uparrow\omega$ comme des fonctions de type $([D \to D] \to D)$, où D est un ensemble ordonné inductif quelconque. En effet, ces opérateurs associent à toute transformation continue f de D un élément $f\uparrow n$ ou $f\uparrow\omega$ appartenant à D. On a à ce propos le théorème suivant.

Théorème 5.11. Les opérateurs $\uparrow n$ $(n \in \mathbf{N})$ et $\uparrow\omega$ sont continus et l'ensemble $\{\uparrow n : n \in \mathbf{N}\}$ est une chaîne dans l'ensemble ordonné inductif $[[D \to D] \to D]$; sa meilleure borne supérieure est $\uparrow\omega$.

Démonstration. On prouve d'abord la continuité des opérateurs $\uparrow n$, par récurrence sur n. Soit $\{f_i : i \in I\}$ une chaîne de l'ensemble $[D \to D]$. On doit prouver l'égalité $(\bigsqcup_i f_i)\uparrow n = \bigsqcup_i (f_i\uparrow n)$. Pour $n = 0$, cette égalité se réduit à $\bot = \bot$. Avant de passer au cas inductif, signalons deux lemmes évidents :

$$m \leq n \;\Rightarrow\; f_i\uparrow m \leq f_i\uparrow n\,,$$
$$f_i \leq f_j \;\Rightarrow\; [f_i(f_i\uparrow n) \leq f_i(f_j\uparrow n) \wedge f_i(f_j\uparrow n) \leq f_j(f_j\uparrow n)]\,.$$

Si on suppose l'égalité $(\bigsqcup_i f_i)\uparrow n = \bigsqcup_i (f_i\uparrow n)$ vraie pour un certain n, on peut écrire

$$\begin{aligned}
(\textstyle\bigsqcup_i f_i)\uparrow(n+1) &= (\textstyle\bigsqcup_i f_i)((\bigsqcup_j f_j)\uparrow n) && [\text{définition de } \uparrow]\\
&= (\textstyle\bigsqcup_i f_i)(\bigsqcup_j (f_j\uparrow n)) && [\text{hypothèse d'induction}]\\
&= \textstyle\bigsqcup_i f_i(\bigsqcup_j (f_j\uparrow n)) && [\text{définition de } \textstyle\bigsqcup_i f_i]\\
&= \textstyle\bigsqcup_i (\bigsqcup_j f_i(f_j\uparrow n)) && [\text{continuité des } f_i]\\
&= \textstyle\bigsqcup_{i,j} f_i(f_j\uparrow n) && [\text{propriété de } \textstyle\bigsqcup]\\
&= \textstyle\bigsqcup_i f_i(f_i\uparrow n) && [\text{lemme préliminaire}]\\
&= \textstyle\bigsqcup_i (f_i\uparrow(n+1)) && [\text{définition de } \uparrow]\,.
\end{aligned}$$

L'égalité est vraie pour $n+1$, et pour tout n. On en déduit

$$\textstyle\bigsqcup_n (\bigsqcup_i f_i)\uparrow n = \bigsqcup_n (\bigsqcup_i (f_i\uparrow n))\,.$$

Le premier membre vaut $(\bigsqcup_i f_i)\uparrow\omega$; en intervertissant les deux opérateurs $\bigsqcup$, le second membre vaut $\bigsqcup_i (f_i\uparrow\omega)$, ce qui établit la continuité de $\uparrow\omega$.

Vu le premier lemme préliminaire, l'ensemble $\{\uparrow n : n \in \mathbf{N}\}$ est une chaîne de l'espace $[[D \to D] \to D]$. La meilleure borne supérieure de cette chaîne applique f sur $\bigsqcup_n (f\uparrow n)$; c'est donc bien $\uparrow\omega$.

Remarque. Une égalité comme $\bigsqcup_i \bigsqcup_j a_i^j = \bigsqcup_j \bigsqcup_i a_i^j$ s'établit systématiquement comme suit, en deux étapes symétriques, dont la première est :

$$a_i^j \leq \textstyle\bigsqcup_j \bigsqcup_i a_i^j\,,$$
$$\textstyle\bigsqcup_j a_i^j \leq \bigsqcup_j \bigsqcup_i a_i^j\,,$$
$$\textstyle\bigsqcup_i \bigsqcup_j a_i^j \leq \bigsqcup_j \bigsqcup_i a_i^j\,.$$

On utilise les deux propriétés caractérisant $\bigsqcup$; tout d'abord, $x_k \leq \bigsqcup_i x_i$, et ensuite, $x_k \leq y$ pour tout k implique $\bigsqcup_k x_k \leq y$.

Dans le même ordre d'idée, on montre aisément la continuité de l'opérateur d'application Ap, qui associe au couple (f, x) la valeur $f(x)$:

Théorème 5.12. Soient D et E deux ensembles ordonnés inductifs; l'opérateur $Ap : (f, x) \to f(x)$ de domaine $[D \to E] \times D$ et de codomaine inclus dans E est continu.

Démonstration. Une chaîne de $[D \to E] \times D$ est du type $\{(f_i, x_i) : i \in I\}$, où $\{f_i\}$ et $\{x_i\}$ sont respectivement des chaînes dans $[D \to E]$ et D. Si les bornes supérieures de ces chaînes sont respectivement f et x, on doit prouver $Ap(\bigsqcup_i(f_i, x_i)) = \bigsqcup_i Ap(f_i, x_i)$, c'est-à-dire

$$f(x) = \bigsqcup_i f_i(x_i)\,.$$

Si $(f_i, x_i) \leq (f_j, x_j)$, on a $f_i(x_i) \leq f_i(x_j) \leq f_j(x_j)$, d'où on déduit

$$\bigsqcup_i f_i(x_i) = \bigsqcup_i \bigsqcup_j f_i(x_j)\,.$$

Vu la continuité de f_i, on a

$$\bigsqcup_j f_i(x_j) = f_i(\bigsqcup_j x_j) = f_i(x)\,,$$

et aussi

$$\bigsqcup_i \bigsqcup_j f_i(x_j) = \bigsqcup_i f_i(x)\,.$$

Enfin, on a

$$\bigsqcup_i f_i(x) = f(x)\,,$$

puisque $f = \bigsqcup_i f_i$, ce qui achève la démonstration.

5.2.6. *Continuité en algèbre, en analyse et en topologie*

La notion algébrique de continuité qui vient d'être introduite semble en contradiction avec la notion classique de continuité utilisée en analyse mathématique. En particulier, les fonctions continues de l'analyse ne sont pas nécessairement croissantes. Néanmoins, l'intuition et la motivation du concept de continuité sont les mêmes en algèbre et en analyse: une fonction f est continue si elle permute avec l'opérateur de passage à la limite. D'une manière plus imagée, pour connaître une bonne approximation y_0 de $f(x)$, il suffit de connaître une bonne approximation x_0 de x, et de choisir $y_0 = f(x_0)$.

Le concept de continuité joue un rôle essentiel dans la suite de ce chapitre; il nous a donc paru souhaitable de l'étudier plus complètement. (Le lecteur non mathématicien peut passer directement au paragraphe 5.2.7.) Le cadre naturel de cette étude est la *topologie* (§ I.9.4.6), dont l'objet est précisément la continuité sous toutes ses formes. Rappelons brièvement quelques définitions.

Un *espace topologique* $(E, \mathcal{O})$ est un ensemble E muni d'une *topologie* $\mathcal{O}$, c'est-à-dire d'une famille de sous-ensembles de E, appelés *ouverts*, vérifiant les propriétés suivantes :

- la réunion d'un ensemble quelconque d'ouverts est un ouvert;
- l'intersection d'un ensemble fini d'ouverts est un ouvert.

Remarques. Quand la topologie est fixée, on écrit E au lieu de $(E, \mathcal{O})$.
Les topologies sur E sont des ensembles particuliers de sous-ensembles de E. La topologie minimale, la plus *grossière*, se compose des deux ouverts E et $\emptyset$. La topologie maximale, la plus *fine*, est l'ensemble $\mathcal{P}(E)$ lui-même : tout sous-ensemble de E est un ouvert.

Les *fermés* sont les complémentaires des ouverts : B est fermé si $E \setminus B$ est ouvert. La réunion d'un ensemble fini de fermés est un fermé et l'intersection d'un ensemble quelconque de fermés est un fermé. L'ensemble $\mathcal{T}$ des fermés caractérise la topologie.[7]

Une fonction f de E dans E' est *continue* si l'image inverse $f^{-1}(A')$ d'un ouvert A' de E' est un ouvert de E :

$$\forall A' \, [A' \in \mathcal{O}' \Rightarrow f^{-1}(A') \in \mathcal{O}] \, .$$

La notion de continuité dépend donc des topologies dont sont munis les deux espaces E et E'. Plus la topologie de E est grossière et plus la topologie de E' est fine, plus la propriété de continuité sera restrictive.

On peut aussi définir la continuité par les fermés : une fonction est continue si et seulement si l'image inverse d'un fermé est toujours un fermé.

La topologie usuelle dans $\mathbf{R}$ est basée sur les intervalles ouverts classiques (du type $]a, b[= \{x : a < x < b\}$). Un ouvert de $\mathbf{R}$ est la réunion d'une famille quelconque d'intervalles ouverts.

5.2.6.1. *De la continuité algébrique à la continuité topologique*

Il est très facile de passer de la continuité algébrique (et d'abord de la croissance, qui en est une forme affaiblie) à la continuité topologique. Etant donné deux ensembles ordonnés $(E_1, \leq)$ et $(E_2, \sqsubseteq)$, on doit construire deux topologies $\mathcal{O}_1$ et $\mathcal{O}_2$ telles que les fonctions croissantes de $(E_1, \leq)$ vers $(E_2, \sqsubseteq)$ s'identifient aux fonctions continues de $(E_1, \mathcal{O}_1)$ sur $(E_2, \mathcal{O}_2)$.

Si un ensemble E est ordonné par une relation ρ, la *topologie d'Alexandrov* sur (E, ρ) est l'ensemble des filtres de (E, ρ); les ouverts sont donc les filtres, et les fermés sont les idéaux.[8] On a le théorème suivant.

Théorème 5.13. La fonction f est croissante de $(E_1, \leq)$ vers $(E_2, \sqsubseteq)$ si et seulement si elle est continue de $(E_1, \mathcal{O}_1)$ vers $(E_2, \mathcal{O}_2)$, où $\mathcal{O}_1$ et $\mathcal{O}_2$ sont les topologies d'Alexandrov sur E_1 et E_2, associées respectivement aux relations d'ordre $\leq$ et $\sqsubseteq$.

Démonstration. La condition est nécessaire. Soit f une fonction croissante et soit I un idéal de $(E_2, \sqsubseteq)$. Il faut montrer que $f^{-1}(I)$ est un idéal de $(E_1, \leq)$, c'est-à-dire que si $x \in f^{-1}(I)$ et $y \leq x$, alors $y \in f^{-1}(I)$. Comme $y \leq x$, on a

[7] Dans le volume I, l'ensemble $\mathcal{T}$ *définissait* la topologie.

[8] Rappelons qu'un filtre est un ensemble contenant les majorants de ses éléments; un idéal contient les minorants de ses éléments.

$f(y) \sqsubseteq f(x)$; comme $f(x) \in I$, on a $f(y) \in I$ et $y \in f^{-1}(I)$.
La condition est suffisante. Soit f une fonction continue pour la topologie d'Alexandrov, $y \in E$ et $I = \downarrow f(y)$. Comme $f^{-1}(I)$ est un idéal contenant y, on a $x \in f^{-1}(I)$ pour tout $x \leq y$, ou encore $f(x) \in I$, ou encore $f(x) \sqsubseteq f(y)$.

On introduit aussi la *topologie de Scott*, moins fine que celle d'Alexandrov. Dans la topologie de Scott, un idéal $I \subset E$ est un fermé si le supremum (éventuel) d'une chaîne incluse dans I est un maximum, donc appartient à I. Dans le cas particulier fréquent des ensembles ordonnés inductifs, un idéal est un fermé (pour la topologie de Scott) s'il contient les meilleures bornes supérieures de ses chaînes; ceci donne lieu à un théorème analogue au précédent.

Théorème 5.14. La fonction f est continue de $(E_1, \leq)$ vers $(E_2, \sqsubseteq)$ si et seulement si elle est continue de $(E_1, \mathcal{O}_1)$ vers $(E_2, \mathcal{O}_2)$, où $\mathcal{O}_1$ et $\mathcal{O}_2$ sont les topologies de Scott sur E_1 et E_2, associées respectivement aux relations d'ordre $\leq$ et $\sqsubseteq$.
Démonstration. La condition est nécessaire. Soit f une fonction continue de $(E_1, \leq)$ vers $(E_2, \sqsubseteq)$ et soit I un idéal de $(E_2, \sqsubseteq)$ comportant les meilleures bornes supérieures de ses chaînes. Il faut montrer que $f^{-1}(I)$ est aussi un idéal et qu'il comporte les meilleures bornes supérieures de ses chaînes. Le premier point est une conséquence du théorème précédent, puisque toute fonction algébriquement continue est croissante. Soit alors C une chaîne de $f^{-1}(I)$; $f(C)$ est une chaîne de I et, comme on a $f(\bigsqcup C) = \bigsqcup f(C)$, on a bien $f(\bigsqcup C) \in I$, et donc $\bigsqcup C \in f^{-1}(I)$.
La condition est suffisante. Soit f une fonction continue pour la topologie de Scott et soit C une chaîne de E. Comme E est inductif, on a $\bigsqcup C \in E$. Soit $I = \downarrow \bigsqcup f(C)$. Comme $f^{-1}(I)$ est un idéal contenant C, il contient aussi la limite $\bigsqcup C$, d'où $f(\bigsqcup C) \in I$, et $f(\bigsqcup C) \sqsubseteq \bigsqcup f(C)$. L'inégalité inverse a lieu si f est croissante. L'argumentation donnée pour la topologie d'Alexandrov (condition suffisante du théorème précédent) reste d'application ici car, dans un espace inductif, un idéal déterminé par un point contient toujours les meilleures bornes supérieures de ses chaînes.

5.2.6.2. *De la continuité topologique à la continuité algébrique*

Les théorèmes précédents ne permettent généralement pas de prouver directement qu'une fonction continue au sens topologique l'est aussi au sens algébrique. La raison est assez claire: à toute relation d'ordre sont associées une topologie d'Alexandrov et une topologie de Scott, mais toute topologie n'est pas la topologie d'Alexandrov ou de Scott associée à une certaine relation d'ordre.

Un exemple classique est celui de la droite réelle. On observe d'abord que, pour l'ordre numérique habituel, l'ensemble $\mathbf{R}$ n'est pas inductif: une suite croissante n'est convergente que si elle est bornée supérieurement. Ce problème

se règle facilement, par adjonction d'un élément $+\infty$, défini comme la limite commune de toutes les suites croissantes non bornées. On constate ensuite que la condition $\lim f(x_n) = f(\lim x_n)$, supposée vérifiée pour toute suite croissante, ne suffit pas à garantir la continuité de f; la fonction qui à x associe le plus petit entier $n \geq x$ est un contre-exemple. Enfin, il est clair que changer la relation d'ordre sur $\mathbf{R}$ ne permettra pas de résoudre ce problème.

La solution s'obtient en retournant à l'interprétation intuitive de la continuité : pour obtenir une approximation de plus en plus précise de $f(x)$, il suffit de disposer d'une approximation de plus en plus précise de x. Formellement, une approximation de x est un ensemble contenant x; une approximation A est plus précise qu'une approximation B si elle est incluse dans B. Une relation d'ordre apparaît clairement, mais elle est définie sur les sous-ensembles de $\mathbf{R}$ plutôt que sur les éléments de $\mathbf{R}$. Nous interpréterons donc la fonction f sur des ensembles de points plutôt que sur des points. Plus précisément, nous considérerons la fonction f' dont le domaine est l'ensemble $\mathcal{A}$ des approximations sur $\mathbf{R}$, définie par $f'(A) =_{def} \{f(x) : x \in A\}$, pour tout $A \in \mathcal{A}$. La démarche naturelle est de prouver que f est continue (pour la topologie usuelle) si et seulement si f' est algébriquement continue sur l'ensemble $(\mathcal{A}, \preceq)$, où $\preceq$ est la relation d'ordre "moins précis que", c'est-à-dire la réciproque de l'inclusion.

Il faut toutefois définir formellement l'ensemble $\mathcal{A}$ (de préférence, sans se limiter au cas où l'espace topologique D de départ est $\mathbf{R}$); il faut aussi montrer que $\mathcal{A}$, muni de la topologie d'Alexandrov relative à $\preceq$, admet D comme sous-espace topologique. Les approximations les plus fines se réduisent à des points. Une approximation moins fine sera un ensemble contenant plus d'un point. Quelle restriction faut-il apporter aux ensembles de ce type ? Dans $\mathbf{R}$, une approximation est un intervalle fermé borné, et un célèbre théorème dû à Cantor affirme que $\bigcap_n I_n$ est un intervalle fermé, borné et non vide si (I_n) est une suite d'intervalles fermés, bornés, emboîtés et non vides; on en déduit (pour plus de détail, voir Livercy 1978) que l'égalité $\bigcap_n f(I_n) = f(\bigcap_n I_n) \neq \emptyset$, exprimant la continuité algébrique de f, exprime aussi la continuité topologique de f. D'une manière (légèrement) plus générale, sur un espace métrique quelconque, on peut assimiler les approximations aux ensembles compacts. Cette assimilation respecte une exigence naturelle : si A est une approximation de x, $f(A)$ est une approximation de $f(x)$, si f est continue. En effet, l'image d'un compact par une fonction continue est compacte. Rappelons ici l'une des deux définitions habituelles de la compacité (§ I.9.4.6) : un sous-ensemble A de l'espace topologique E est *compact* si pour toute famille d'ouverts $\mathcal{O} = \{O_i : i \in I\}$ telle que $A \subset \bigcup \mathcal{O}$, il existe une sous-famille finie $\mathcal{O}'$ telle que $A \subset \bigcup \mathcal{O}'$. Nous noterons $\mathcal{H}_E$ l'ensemble des compacts d'un espace topologique E. Remarquons que pour tout $x \in E$, on a $\{x\} \in \mathcal{H}_E$, ce qui veut dire que E est (assimilable à) un sous-espace de $\mathcal{H}_E$. Plus concrètement, un point est une approximation particulière (de précision totale). Si f est une fonction continue de E dans F, f s'étend immédiatement en une fonction f'

de $\mathcal{H}_E$ dans $\mathcal{H}_F$. Sinon, l'image $f(A)$ n'est pas nécessairement un compact, mais on peut poser $f'(A) = \bigcap\{B \in \mathcal{H}_F : f(A) \subset B\}$, c'est-à-dire, $f'(A)$ est le plus petit compact contenant $f(A)$. (Si aucun compact ne contient $f(A)$, on pose $f'(A) = F$.) Le théorème suivant résume ces considérations.

Théorème 5.15. La fonction $f : E \to F$ est continue au sens de la topologie si et seulement si la fonction $f' : \mathcal{H}_E \to \mathcal{H}_F$ est continue au sens de l'algèbre.

On peut aussi munir les espaces $\mathcal{H}_E$ et $\mathcal{H}_F$ de métriques compatibles avec celles de E et F. Pour ce faire, il suffit de généraliser la notion de distance entre deux points en une notion de distance entre deux compacts. L'approche naïve qui définirait $d'(A,B) = inf\{d(x,y) : x \in A \wedge y \in B\}$ ne conviendrait pas ... parce que d' ne serait pas une distance; en particulier, $d'(A,B) = 0$ n'impliquerait pas $A = B$. Nous utiliserons quand même la notation d', en sachant qu'il ne s'agit pas d'une distance.

Pour tout A et pour tout $\varepsilon \geq 0$ on pose $A + \varepsilon = \{x : d(x,A) \leq \varepsilon\}$; il s'agit de l'ensemble des points "proches" de A. On obtient alors la distance (de Hausdorff) comme suit : $h(A,B) \leq \varepsilon$ si et seulement si $A \subset B + \varepsilon$ et $B \subset A + \varepsilon$. On vérifie aisément que h est une distance sur $\mathcal{H}_E$, compatible avec la distance d : $h(\{x\},\{y\}) = d(x,y)$, pour tous $x, y \in E$. "La boucle est bouclée" avec le théorème suivant.

Théorème 5.16. La fonction $f : E \to F$ est continue si et seulement si son extension $f' : \mathcal{H}_E \to \mathcal{H}_F$ est continue.

Dans les deux cas, il s'agit de continuité au sens de la topologie; dans le premier cas, les métriques sont d_E et d_F, dans le second cas, on utilise les métriques de Hausdorff h_E et h_F.

5.2.7. *Points fixes dans un treillis*

Il arrive fréquemment que la recherche de points fixes se fasse dans un treillis, et non dans un ensemble ordonné quelconque. Un *treillis* est un ensemble ordonné dans lequel toute paire (et donc, tout ensemble fini) admet une meilleure borne inférieure et une meilleure borne supérieure.

Remarques. En algèbre, un treillis est parfois défini comme un ensemble muni de deux opérations binaires $\vee$ et $\wedge$ jouissant des propriétés classiques d'associativité, de commutativité et d'idempotence, et de la propriété dite d'absorption : $x \wedge (x \vee y) = x = x \vee (x \wedge y)$. Dans ce cas, la notion d'ordre est dérivée; $x \leq y$ est par définition $x = x \wedge y$ (ou $x \vee y = y$).

La notion générique d'idéal est rendue plus restrictive dans un treillis.[9] Un *idéal* (de treillis) contient non seulement les minorants de ses éléments, mais aussi

[9] C'est dans le cadre des treillis que la notion d'idéal et celle, duale, de filtre, sont le plus souvent évoquées dans la littérature. Ces notions sont aussi importantes en topologie.

les meilleures bornes supérieures des paires de ses éléments. Cette restriction n'altère pas le résultat suivant : l'ensemble $\downarrow x$ des minorants de x est l'idéal principal engendré par x (voir la note accompagnant le théorème 3.3). Il existe une notion duale : un *filtre* contient les majorants de ses éléments et les plus grandes bornes inférieures des paires de ses éléments.

Un treillis E est *complet* si tout sous-ensemble de E admet un supremum. Un treillis complet est nécessairement inductif; de plus, on a le théorème suivant, qui justifie l'emploi du mot "complet".

Théorème 5.17. Dans un treillis complet, tout sous-ensemble admet une meilleure borne supérieure et une meilleure borne inférieure.

Démonstration. La meilleure borne inférieure du sous-ensemble A du treillis complet E est la meilleure borne supérieure de l'ensemble des minorants de A.

Exemples. Tout treillis fini est complet. Le treillis formé par les sous-ensembles d'un ensemble E quelconque est complet pour la relation d'inclusion. En logique, on définit des treillis de formules, où les opérations $\wedge$ et $\vee$ sont naturellement la conjonction et la disjonction, tandis que la relation d'ordre est l'implication; ces treillis ne sont pas toujours complets.

Théorème 5.18. Si f est une fonction croissante dont le domaine est un treillis complet E et dont le codomaine est inclus dans E, alors le plus petit point fixe de f est $\bigsqcap\{x \in E : f(x) \leq x\}$ et le plus grand point fixe de f est $\bigsqcup\{x \in E : x \leq f(x)\}$.

C'est un corollaire des théorèmes précédents.

Remarque. Il existe une réciproque, dont la démonstration est moins évidente : un treillis dans lequel toute fonction croissante admet un point fixe est complet.

Dans un treillis complet E, il existe non seulement un minimum, noté $\bot$, mais aussi un maximum, noté $\top$. Si f est une transformation croissante de E, on pose

$$f{\downarrow}0 = \top\,,$$
$$f{\downarrow}r' = f(f{\downarrow}r)\,, \text{pour tout ordinal } r\,,$$
$$f{\downarrow}\ell = \bigsqcap\{f{\downarrow}\alpha : \alpha \in \ell\}\,, \text{pour tout ordinal limite } \ell\,.$$

Avec ces notations, il existe un plus petit ordinal α et un plus petit ordinal β tel que le plus petit point fixe de f soit $f{\uparrow}\alpha$ et que le plus grand point fixe de f soit $f{\downarrow}\beta$. Si la fonction f est continue, alors α est au plus égal à ω (mais β peut être plus grand que ω).

Remarque. Historiquement, la notion de point fixe a d'abord été développée dans le cadre des treillis. Un certain temps s'est écoulé avant l'utilisation de cette notion dans le cadre des ensembles ordonnés inductifs. Ce cadre est pourtant plus général et, dans la majeure partie des cas, mieux adapté aux applications informatiques. Un des premiers résultats significatifs est le théorème 5.18, dû à Knaster et Tarski (sous une forme un peu plus faible); la théorie du point fixe a été développée en grande partie par Scott.

5.2.8. *Exemples classiques*

Si E est un ensemble quelconque, alors $(\mathcal{P}(E), \subset)$ est un treillis complet. La meilleure borne supérieure d'un ensemble de parties de E est sa réunion et sa meilleure borne inférieure est son intersection. (Dans ce contexte, on pose naturellement $\bigcap \emptyset = E$.)

L'ensemble Ψ des fonctions partielles de E dans F est naturellement ordonné par inclusion (rappelons qu'une fonction est, dans le cadre ensembliste, un ensemble de couples). L'ensemble Ψ admet le minimum $\emptyset$, c'est-à-dire la fonction nulle part définie. Les chaînes de Ψ s'interprètent naturellement comme les ensembles d'approximations de fonctions. Si C est une chaîne, alors pour toutes fonctions $f, g \in C$ on a $f \subset g$ ou $g \subset f$; $f \subset g$ signifie que, pour tout $x \in dom(f)$, on a $f(x) = g(x)$.[10] La borne supérieure de la chaîne C est simplement sa réunion, c'est-à-dire la fonction h telle que $h(x) = a$ s'il existe $f \in C$ telle que $f(x) = a$. On en déduit que Ψ est un ensemble ordonné inductif.

Si l'ensemble E est muni d'une topologie, l'ensemble $\mathcal{O}$ des ouverts de E est un treillis complet, si on pose

$$\bigsqcup A = \bigcup A\,,\ \bigsqcap A = (\bigcap A)^{\circ}\,.$$

L'ensemble $\mathcal{T}$ des fermés de E est également un treillis complet, avec

$$\bigsqcup A = \overline{\bigcup A}\,,\ \bigsqcap A = \bigcap A\,.$$

Dans les deux cas, la relation d'ordre sous-jacente est naturellement l'inclusion. Rappelons que X° est l'*intérieur* de X, c'est-à-dire le plus grand ouvert (la réunion de tous les ouverts) inclus dans X; de même, $\overline{X}$ est l'*adhérence* de X, c'est-à-dire le plus petit fermé (l'intersection de tous les fermés) contenant X.

5.2.9. *Propriétés des points fixes*

Les paragraphes précédents fournissent quelques moyens de calcul de points fixes, et notamment de plus petits points fixes. Les quelques exemples déjà rencontrés montrent que les points fixes n'héritent généralement pas des propriétés des éléments des chaînes dont ils sont les meilleures bornes supérieures. Ce phénomène est d'ailleurs commun à la plupart des processus d'approximation. Par exemple, $\sqrt{2}$ est limite de rationnels sans être rationnel; la fonction e^x est limite de polynômes sans être un polynôme, etc.

Des critères sont donc nécessaires pour déterminer quelles propriétés sont préservées lors du "passage à la limite". Le théorème de Scott montre que la relation d'ordre est préservée par passage à la limite.

---

[10] Nous avons évoqué au paragraphe 2.2.3 la possibilité d'étendre le domaine d'une fonction (ou de rendre totale sur un ensemble E une fonction partielle) en posant $f(x) = \perp$ pour tout $x \in E \setminus dom(f)$. Le choix du symbole $\perp$ pour désigner la valeur conventionnelle "non défini" n'est pas un hasard; on reviendra sur ce point au paragraphe 5.3.

Théorème 5.19. Soient les ensembles ordonnés inductifs D et E et les applications continues $f : E \to E$, $g_1 : E \to D$ et $g_2 : E \to D$. Si on a

$$g_1(\bot) \leq g_2(\bot),$$
$$(\forall a \in E)\,(g_1(a) \leq g_2(a) \Rightarrow g_1(f(a)) \leq g_2(f(a))),$$

alors on a aussi

$$g_1(f{\uparrow}\omega) \leq g_2(f{\uparrow}\omega).$$

Démonstration. On établit facilement par induction que $g_1(f{\uparrow}n) \leq g_2(f{\uparrow}n)$, pour tout $n \in \mathbf{N}$ d'où, vu la continuité de g_1 et g_2, on a $g_1(\bigsqcup f{\uparrow}n) \leq g_2(\bigsqcup f{\uparrow}n)$. Comme la fonction f est continue, on a aussi $g_1(f{\uparrow}\omega) \leq g_2(f{\uparrow}\omega)$, puisque $f{\uparrow}\omega = \bigsqcup f{\uparrow}n$.

Une propriété *admissible* est une propriété à laquelle le théorème de Scott s'applique. Plus précisément, la propriété P définie sur les éléments de E est admissible s'il existe deux familles de fonctions $f_i, g_i : E \to D$, $i \in I$, telles que

$$(\forall x \in E)\,(P(x) \equiv \forall i\,[f_i(x) \leq g_i(x)]).$$

Le théorème de Park donne un moyen de comparer le plus petit point fixe d'une fonction croissante à certains éléments du domaine.

Théorème 5.20. Soit D un ensemble ordonné inductif et $f : D \to D$ une fonction croissante. Si pour un élément $a \in D$ on a

$$f(a) \leq a,$$

alors on a aussi

$$d \leq a,$$

où d est le plus petit point fixe de f.

Démonstration. Le théorème 5.5 permet d'écrire

$$d = \bigsqcap\{x \in D : f(x) \leq x\};$$

a est donc un élément d'un ensemble dont d est le minimum.

Remarque. La réciproque est fausse. On peut avoir $d \leq a$ sans avoir $f(a) \leq a$. Un contre-exemple est $f = \{(0,0),(1,2),(2,2)\}$, avec la relation d'ordre arithmétique, $d = 0$ et $a = 1$.

5.3. Application à la programmation fonctionnelle

La plupart des programmes informatiques ont pour but de calculer des *résultats* en fonction de *données*. On modélise naturellement les programmes de ce type par des fonctions, qui appliquent les données sur les résultats correspondants.

La programmation fonctionnelle est née d'une observation élémentaire : si la sémantique d'un programme est une fonction, il est possible, et même naturel, de programmer au moyen de fonctions. La majorité des langages algorithmiques actuellement utilisés font d'ailleurs une place plus ou moins importante au style fonctionnel de programmation.

5.3.1. *Introduction*

Le mécanisme de définition de fonctions, tel qu'il est habituellement utilisé, permet de "définir" une fonction f par une phrase du type :

$$f(x) = T(f)(x)\,.$$

On appelle *fonctionnelle* une fonction telle que T, dont le domaine est un ensemble de fonctions.
A titre d'exemple, nous donnons ci-dessous les fonctionnelles T_1 et T_2 respectivement utilisées pour "définir" la fonction factorielle et la fonction plus grand commun diviseur.

$$T_1(f)(x) \;=_{def}\; \text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1)\,.$$

$$\begin{aligned} T_2(g)(x,y) \;=_{def}\; & \text{if } x = y \text{ then } x \\ & \text{else if } x > y \text{ then } g(x-y,y) \\ & \text{else } g(x,y-x)\,. \end{aligned}$$

On vérifie immédiatement que la fonction factorielle est une solution de l'équation $f = T_1(f)$, tandis que la fonction plus grand commun diviseur est une solution de l'équation $g = T_2(g)$. Nous avons mis le mot "définir" entre guillemets parce que rien ne permet a priori de garantir l'existence et l'unicité des fonctions f et g. Puisque les définitions ont la forme d'équations de points fixes, il est naturel d'utiliser la théorie du point fixe pour donner une sémantique, c'est-à-dire une signification précise, à f et g. Encore faudra-t-il s'assurer que cette sémantique permet le calcul effectif, au moyen d'un algorithme d'évaluation à préciser, d'expressions telles que $f(3)$ et $g(12,16)$.

Dans le cas de la définition récursive de la factorielle, on choisit comme première approximation la fonction non définie, ou vide, $f_0 = \emptyset$, et on pose naturellement $f_{n+1} = T_1(f_n)$. On vérifie par récurrence que, pour tout n, on a $f_n = T_1^n(\emptyset) = \{(0,0!),(1,1!),\ldots,(n-1,(n-1)!)\}$,[11] ou encore

$$f_n(x) \;=\; [\text{if } 0 \leq x < n \text{ then } x! \text{ else } \perp]\,,$$

où $\perp$ est une valeur fictive, conventionnellement attribuée à tout argument en dehors du domaine de définition d'une fonction. L'égalité est vraie par hypothèse si $n = 0$. D'autre part, si elle est vraie pour un certain n, alors $f_{n+1}(x) = T_1(f_n)(x)$ vaudra 1 si $x = 0$ et $x * f_n(x-1)$ sinon. On aura donc $f_{n+1}(x) = x*(x-1)!$ si $0 \leq x-1 < n$, et donc $f_{n+1}(x) = x!$ si $0 \leq x < n+1$; enfin, si $x > n$, alors $f_{n+1}(x)$ sera non défini, comme $f_n(x)$. On peut également vérifier que la factorielle est la seule solution de l'équation $f = T_1(f)$, par application du schéma de von Neumann. Si une autre fonction φ, distincte de la factorielle, était solution de l'équation, l'ensemble des valeurs discriminant la fonction φ et la factorielle serait non vide et admettrait un minimum m,

[11] Comme au chapitre 2, nous assimilons une fonction à un ensemble de couples; rappelons aussi que, pour toute transformation ϕ, ϕ^0 est la relation identique, et $\phi^{n+1} = \phi \circ \phi^n$.

nécessairement distinct de 0; de $\varphi(m-1) = (m-1)!$ on déduit $\varphi(m) = m!$, d'où m n'existe pas.

Dans le cas de l'équation $g = T_2(g)$, le fait que la fonction plus grand commun diviseur soit une solution de l'équation est une conséquence des propriétés classiques de cette fonction. En particulier, on a $pgcd(x,x) = x$ et, si $x > y$, alors $pgcd(x,y) = pgcd(x-y,y)$. On prouve par l'absurde, comme précédemment, que *pgcd* est la seule solution de l'équation $g = T_2(g)$. Si la solution g' est distincte de la solution *pgcd*, l'ensemble E des couples (x,y) d'entiers strictement positifs tels que $g'(x,y) \neq pgcd(x,y)$ est non vide. L'ensemble $\mathbf{N}_0 \times \mathbf{N}_0$ est muni d'une relation d'ordre bien fondé: on pose $(x_1,y_1) < (x_2,y_2)$ si $x_1 + y_1 < x_2 + y_2$. Soit donc (x_0,y_0) un élément minimal de E. On a $x_0 \neq y_0$, car $g'(x_0,x_0) = x_0 = pgcd(x_0,y_0)$. Supposons donc $x_0 > y_0$. On a $g'(x_0,y_0) = g'(x_0-y_0,y_0) = pgcd(x_0-y_0,y_0) = pgcd(x_0,y_0)$, car $(x_0-y_0,y_0) < (x_0,y_0)$, d'où $(x_0,y_0) \notin E$.

Le but premier de cette section sera de systématiser et de rendre rigoureuse la définition récursive des fonctions. Une approche informelle ne serait pas satisfaisante, parce que les cas "pathologiques" sont nombreux. Il est clair notamment que les fonctionnelles T_3 et T_4, respectivement définies par

$$T_3(f)(x) = f(x)\,,$$
$$T_4(f)(x) = f(x) + 1\,,$$

ne "définissent" pas de fonctions intéressantes. En effet, toutes les fonctions vérifient l'équation $f = T_3(f)$, tandis qu'aucune (sauf éventuellement la fonction vide, si on donne un sens à une égalité du type $\perp = \perp + 1$) ne vérifie l'équation $f = T_4(f)$.

5.3.2. *Ensembles étendus*

Si nous voulons utiliser la théorie du point fixe pour justifier le principe de définition récursive de fonction, la première étape est de munir les ensembles de fonctions d'une relation d'ordre appropriée, que nous noterons $\sqsubseteq$. Dans le cas de la définition récursive de la fonction factorielle, et dans tous les cas pour lesquels la méthode des approximations successives fonctionne bien, on devra avoir

$$\emptyset = T{\uparrow}0 \sqsubseteq T(\emptyset) = T{\uparrow}1 \sqsubseteq T^2(\emptyset) = T{\uparrow}2 \sqsubseteq T^3(\emptyset) = T{\uparrow}3 \sqsubseteq \cdots,$$

où T est la fonctionnelle définissante. L'exemple de la factorielle suggère immédiatement que la relation $\sqsubseteq$ n'est autre que l'inclusion, si on assimile une fonction à un ensemble de couples.

En programmation fonctionnelle, on définit non seulement des fonctions ordinaires, mais aussi des fonctions dont certains arguments peuvent être des fonctions. Donnons deux exemples classiques. Etant donné une liste L de nombres entiers (le type de L sera l'ensemble $\mathbf{Z}^*$) et une fonction $f : \mathbf{Z} \to \mathbf{Z}$, désignons par $[f : L]$ la liste obtenue en appliquant f à chaque élément de L. Il

est assez naturel de vouloir définir une fonction $\mu : ([\mathbf{Z} \to \mathbf{Z}] \times \mathbf{Z}^*) \to \mathbf{Z}^*$ telle que $\mu(f, L) = [f : L]$. Un autre exemple est celui des procédures génériques de tri, qui prennent comme arguments une liste d'éléments d'un ensemble D et une relation de comparaison sur cet ensemble. Ceci montre qu'il faudra définir la relation d'ordre $\sqsubseteq$ non seulement sur les ensembles fonctionnels de type $(\mathbf{Z} \to \mathbf{Z})$, mais aussi sur des ensembles plus complexes.

Les ensembles qui nous intéressent ici seront d'une part les ensembles primitifs ($\mathbf{N}$ et $\mathbf{Z}$ par exemple) et d'autre part des ensembles dérivés au moyen d'opérations comme le produit cartésien $\times$ ou la "fonctorisation" $\to$. Si D et E sont des ensembles pertinents, on s'attend à ce que $D \times E$ et $D \to E$ le soient aussi.

L'exemple de la factorielle suggère que $\sqsubseteq$ peut être assimilé à l'inclusion, sur des ensembles fonctionnels ordinaires. Il y a cependant un problème: les ensembles de couples constituant les approximations successives de la factorielle ne sont pas des applications (fonctions totales), éléments de $(\mathbf{N} \to \mathbf{N})$, mais des fonctions partielles. Le même exemple suggère le remède: une fonction partielle de $\mathbf{N}$ dans $\mathbf{N}$ est une fonction totale de $\mathbf{N}$ dans $\mathbf{N}^+$, où $\mathbf{N}^+$ est simplement $\mathbf{N}$ enrichi de l'élément fictif "non défini", que nous avons noté $\perp$. Avec cette convention, $f \sqsubseteq g$ signifie simplement que, pour tout naturel n, on a $f(n) = g(n)$ ou $f(n) = \perp$. Autrement dit, f est "moins défini ou égal" à g.

La notion d'ordre canonique induit (§ 5.2.5) permet de généraliser une relation d'ordre sur E en une relation d'ordre sur $(D \to E)$. On peut donc se demander quelle relation d'ordre sur $\mathbf{N}^+$ (également notée $\sqsubseteq$) a pu induire la relation d'ordre $\sqsubseteq$ sur l'ensemble fonctionnel $(\mathbf{N} \to \mathbf{N}^+)$; la réponse est immédiate: dans $\mathbf{N}^+$, $n \sqsubseteq m$ signifie $n = m \vee n = \perp$.[12]

Ceci se généralise immédiatement. On convient d'ajouter à chaque ensemble primitif D (supposé non ordonné),[13] un nouvel élément désigné par $\perp$ (*bottom*) servant de minimum. Il sert aussi de "résultat fictif" à une opération non définie, ce qui justifie une notation commune $\perp$ pour les notions "non défini" et "minimum". L'ensemble étendu ainsi défini est noté D^+.

Le passage des ensembles primitifs aux ensembles étendus rend totales des fonctions qui n'étaient que partielles. Dans le cas de la soustraction naturelle, par exemple, on aura donc $3 - 2 = 1$ et $2 - 3 = \perp$. La soustraction est devenue une opération de $\mathbf{N} \times \mathbf{N}$ dans $\mathbf{N}^+$. Pour permettre la composition des soustractions, par exemple $(2-3)-3 = \perp-3$, il faut étendre cette définition en posant $x-\perp = \perp-x = \perp$, pour tout $x \in \mathbf{N}^+$. La soustraction devient alors une opération (fonction totale) de $\mathbf{N}^+ \times \mathbf{N}^+$ dans $\mathbf{N}^+$.

[12] La notation $\sqsubseteq$ est classiquement employée pour désigner toute relation d'ordre signifiant intuitivement "moins défini ou égal".

[13] L'ensemble $\mathbf{N}$ est naturellement ordonné mais, dans le présent contexte, il n'est pas fait usage de la relation d'ordre numérique.

Remarque. On a d'abord assimilé l'ensemble des fonctions partielles de D dans E à l'ensemble $(D \to E^+)$, formé d'applications (fonctions totales), par *extension d'ensemble.* On est ensuite passé à l'ensemble $(D^+ \to E^+)$ par *extension de fonctions.* Sauf mention explicite du contraire, nous ne considérons que des extensions *naturelles*, ou *strictes*: une *forme*, c'est-à-dire l'application d'une fonction à un nombre adéquat d'arguments, vaudra $\perp$ dès que l'un de ses arguments vaut $\perp$. Une forme ne sera donc définie que si tous ses arguments le sont.

Le domaine des quatre opérations arithmétiques n'est pas primitif, mais obtenu à partir de $\mathbf{N}$ par produit cartésien. L'extension d'un produit cartésien est définie simplement comme le produit cartésien des extensions des facteurs : $(D_1 \times D_2)^+ = D_1^+ \times D_2^+$; la notation $(x_1, x_2) \sqsubseteq (y_1, y_2)$ est équivalente à la notation $(x_1 \sqsubseteq y_1 \wedge x_2 \sqsubseteq y_2)$. Une opération arithmétique est donc un élément de l'ensemble fonctionnel $((\mathbf{N}^+ \times \mathbf{N}^+) \to \mathbf{N}^+)$.

Nous avons introduit une relation d'ordre générique $\sqsubseteq$ sur tout ensemble auquel est susceptible d'appartenir un objet intervenant en programmation fonctionnelle.[14] Ordonner les ensembles de référence n'est toutefois pas suffisant, puisque la théorie du point fixe requiert au minimum la croissance des fonctions auxquelles elle s'applique. L'application est en outre plus aisée si les ensembles ordonnés sont inductifs et si les fonctions sont continues.

On observe d'abord que les chaînes d'un ensemble primitif étendu D^+ comportent au plus deux éléments : $\perp$ et un élément quelconque de D. Les chaînes d'un ensemble étendu obtenu par produit de n facteurs primitifs comportent au plus $n + 1$ éléments et sont donc stationnaires. Un ensemble étendu de ce type sera donc inductif.

Avant de passer au cas moins évident des ensembles fonctionnels, donnons une description informelle mais très éclairante des notions de croissance et de continuité appliquées à la relation $\sqsubseteq$. Si on voit une fonction comme un mécanisme recevant des données et produisant des résultats (c'est le point de vue de l'informaticien), une fonction f est croissante si, en fournissant au mécanisme f une donnée y mieux définie qu'une donnée x, on obtient un résultat $f(y)$ mieux défini que le résultat $f(x)$. Ceci suggère le théorème suivant.

Théorème 5.21. Toute fonction étendue naturellement est croissante.

Démonstration. Si f n'est pas croissante, il existe $a = (a_1, \ldots, a_n)$ et $b = (b_1, \ldots, b_n)$ tels que $a \sqsubseteq b$ et $f(a) \not\sqsubseteq f(b)$; ceci implique qu'il existe un indice i tel que b_i est défini mais non a_i. On doit donc avoir, si l'extension est naturelle, $f(a) = \perp$, ce qui contredit $f(a) \not\sqsubseteq f(b)$.

[14] Ce n'est pas tout à fait exact; on peut concevoir des opérateurs de construction d'ensemble autres que $\times$ et $\to$, la réunion disjointe notamment. On pourrait facilement tenir compte de nouveaux opérateurs.

Remarque. Toutes les chaînes étant stationnaires, les fonctions croissantes sont aussi continues.

La fonction *if-then-else* est par exception étendue non naturellement en un objet de l'ensemble $(\{true, false\}^+ \times D^+ \times D^+) \to D^+$, au moyen des égalités

- (if *true* then a else b) $= a$,
- (if *false* then a else b) $= b$,
- (if $\perp$ then a else b) $= \perp$,

où a et b sont des éléments quelconques de D^+. L'idée est de permettre qu'une branche de l'alternative soit non définie quand l'autre branche est sélectionnée; par exemple, dans la première égalité, on peut avoir $b = \perp$ et $a \neq \perp$. Il est facile de vérifier que cette extension non naturelle est néanmoins croissante.

Revenons à présent aux ensembles fonctionnels. L'exemple des approximations successives de la factorielle engendrées par la fonctionnelle T_1 suggère que les chaînes de fonctions, sans être toujours stationnaires, admettent une meilleure borne supérieure. Il est facile de prouver ceci formellement dans un cas particulier comme la factorielle, ou informellement dans le cas général, en recourant à l'interprétation "moins défini ou égal", attribuée à la relation d'ordre $\sqsubseteq$. Pour obtenir une démonstration formelle et générale, il est tentant de recourir aux théorèmes 5.10 et 5.11; ceux-ci permettent d'appréhender non seulement les ensembles fonctionnels simples, mais les ensembles plus complexes, obtenus par plusieurs applications de l'opérateur de fonctorisation. Le seul obstacle est que ces théorèmes impliquent la restriction aux fonctions (et fonctionnelles) continues. Il convient donc de vérifier d'abord que cette restriction est acceptable ici. Pour ce faire, une interprétation intuitive de la continuité serait utile. Les résultats du paragraphe 5.2.6 suggèrent que l'interprétation intuitive donnée en analyse pourrait être un bon point de départ. Une fonction f est continue si $\lim f(x_n) = f(\lim x_n)$, où (x_n) est une suite convergente quelconque. On pourra notamment obtenir le nombre $\sqrt{\pi}$ comme suite des nombres $\sqrt{3}$, $\sqrt{3.1}$, $\sqrt{3.14}$, etc. Non seulement une meilleure approximation de π donnera une meilleure approximation de $\sqrt{\pi}$ (croissance, au sens de $\sqsubseteq$), mais en plus, pour tout degré d'approximation requis pour $\sqrt{\pi}$, il existera un degré d'approximation adéquat pour π (règle dite ε-δ). Une fonction f est continue si, pour obtenir une approximation adéquate du résultat $f(x)$, il suffit de fournir une approximation adéquate de x. Dans le cas de la factorielle, la fonctionnelle T_1 vérifie cette condition. Pour tout $n \in \mathbf{N}$, il existe $m \in \mathbf{N}$ tel que $n! = T_1{\uparrow}m(n) = T_1^m(\emptyset)(n)$. Il suffit de prendre $m = n + 1$.

L'essence même du principe de la définition récursive en programmation fonctionnelle est la continuité. Si on définit récursivement une fonction f, on ne requiert pas que le nombre d'appels récursifs soit uniformément borné, mais on requiert certainement qu'il soit borné pour chaque donnée x pour laquelle on veut connaître le résultat $f(x)$. Ceci nous conduit à accepter la restriction aux fonctions continues.

Conformément à un usage assez répandu, nous appellerons *domaines* les ensembles ordonnés inductifs pouvant être utilisés en programmation fonctionnelle. Les règles de formation des domaines sont les suivantes.

- Si D est un ensemble non ordonné,[15] $(D^+, \sqsubseteq)$ est un domaine (primitif).
- Si $(E, \sqsubseteq)$ et $(F, \sqsubseteq)$ sont des domaines, alors $(E \times F, \sqsubseteq)$ est un domaine.
- Si $(E, \sqsubseteq)$ et $(F, \sqsubseteq)$ sont des domaines, alors $([E \to F], \sqsubseteq)$ est un domaine. (Les crochets rappellent la restriction aux fonctions continues.)

Rappelons comment la relation d'ordre générique $\sqsubseteq$ est imposée aux domaines primitifs et généralisée aux domaines dérivés.

- $(\forall x, y \in D^+)\,(x \sqsubseteq y \;\equiv\; (x = y \vee x = \bot))$.
- $(\forall (x_1, x_2), (y_1, y_2) \in E \times F)\,((x_1, x_2) \sqsubseteq (y_1, y_2) \;\equiv\; (x_1 \sqsubseteq y_1 \wedge x_2 \sqsubseteq y_2))$.
- $(\forall f, g \in [E \to F])\,(f \sqsubseteq g \;\equiv\; (\forall x \in E)\,(f(x) \sqsubseteq g(x)))$.

Remarques. Certains auteurs emploient le mot "domaine" pour désigner des ensembles ordonnés inductifs munis de propriétés particulières (que possèdent les domaines définis par les règles de formation ci-dessus). Cet usage n'est pas à recommander, parce que le mot "domaine" a déjà plusieurs acceptions en mathématique. En outre, il n'y a pas accord sur les propriétés particulières à imposer. Il est parfaitement possible, par exemple, de développer la "théorie des domaines" dans le cadre des treillis complets plutôt que dans celui des ensembles ordonnés inductifs. Il suffit d'ajouter aux domaines primitifs un élément $\top$ (maximum) interprété comme "inconsistant" ou "surspécifié". Si par exemple $f(x) = 3$ et $g(x) = 4$, on posera $(\bigsqcap\{f, g\})(x) = \bot$ et $(\bigsqcup\{f, g\})(x) = \top$. (Cette approche présente certains avantages, notamment dans le contexte des bases de données, mais nous ne l'avons pas adoptée ici parce que, dans le contexte de la sémantique des programmes, l'objet $\top$ (*top*) n'a pas de signification intuitive, contrairement à l'objet $\bot$.) Rappelons aussi que la liste des règles acceptables pour la formation des domaines ne se restreint pas obligatoirement aux trois règles que nous avons données.

5.3.3. *Programmation fonctionnelle du premier ordre*

Dans ce paragraphe, nous étudions plus concrètement les domaines du type $[D \to E]$, où D et E sont des domaines non fonctionnels. Un élément de ce domaine sera défini comme le plus petit point fixe d'une fonctionnelle (continue) $T : [D \to E] \to [D \to E]$. Les fonctionnelles T_1, T_2, T_3 et T_4 introduites au paragraphe 5.3.1 sont de ce type. Il serait fastidieux de devoir prouver dans chaque cas que la fonctionnelle définissante est bien continue; il est plus commode d'isoler une classe de fonctionnelles englobant toutes (ou presque

[15] Les ensembles ordonnés sont acceptables si la relation d'ordre est ignorée.

toutes) les fonctionnelles utiles, et de prouver que tout élément de cette classe est continu (Manna 1974).

L'exemple habituel

$$f(x) = T_1(f)(x) \quad =_{def} \quad \text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1)$$

suggère les mécanismes à prendre en compte. Une classe $\mathcal{C}$ adéquate est définie par les règles suivantes. Une fonctionnelle appartenant à la classe $\mathcal{C}$ sera dite *bien formée.*

1. Si g est une fonction continue, alors la fonctionnelle constante $T_g : f \to T_g(f) = g$ est bien formée.
2. Si $T_1, \ldots, T_n \in \mathcal{C}$, alors la fonctionnelle $T : f \to T(f) = f(T_1(f), \ldots, T_n(f))$ est bien formée.
3. Si $T_1, \ldots, T_n \in \mathcal{C}$ et si g est une fonction continue, alors la fonctionnelle $S : f \to S(f) = g(T_1(f), \ldots, T_n(f))$ est bien formée.
4. Si $B, T_1, T_2 \in \mathcal{C}$, alors la fonctionnelle $U : f \to U(f) = [\text{if } B(f) \text{ then } T_1(f) \text{ else } T_2(f)]$ est bien formée.

Remarques. Les objets sont supposés de types appropriés. La classe $\mathcal{C}$ contient visiblement des fonctionnelles de types (domaines et codomaines) différents. La fonction g doit avoir pour domaine et pour codomaine des ensembles non fonctionnels. La fonction g sera quasi toujours une extension naturelle, automatiquement continue. D'autres mécanismes pourraient être ajoutés. En particulier, en programmation fonctionnelle du second ordre, on peut composer les fonctionnelles, ce que nous avons exclu ici.

La fonctionnelle T_1 définissant la factorielle est bien formée. C'est une alternative (cas 4) dont les deux premiers arguments sont des fonctionnelles constantes (f n'intervient pas dans $T(f)$, cas 1). Le troisième argument est obtenu par composition non récursive (cas 3) de deux arguments, le premier constant et le second obtenu par composition récursive (cas 2) au départ d'une fonctionnelle constante.

Théorème 5.22. Toute fonctionnelle bien formée est continue.

Démonstration. Il est clair que les fonctionnelles constantes sont continues. D'autre part, si les fonctionnelles $T_1, \ldots, T_n$ sont continues, alors les fonctionnelles T et S, respectivement définies par

$$T(f) = f(T_1(f), \ldots, T_n(f)),$$
$$S(f) = g(T_1(f), \ldots, T_n(f)),$$

où g est une fonction continue (étendue naturellement) de type approprié, sont continues. C'est une application immédiate du théorème des fonctions composées, mais il est intéressant de donner une démonstration directe. Dans le premier cas, on doit avoir

$$\begin{aligned} T(\bigsqcup f_i)(x) &= (\bigsqcup f_i)(T_1(\bigsqcup f_i)(x), \ldots, T_n(\bigsqcup f_i)(x)), \\ &= (\bigsqcup f_i)(\bigsqcup T_1(f_i)(x), \ldots, \bigsqcup T_n(f_i)(x)), \\ &= \bigsqcup f_i(\bigsqcup T_1(f_i)(x), \ldots, \bigsqcup T_n(f_i)(x)), \\ &= \bigsqcup f_i(T_1(f_i)(x), \ldots, T_n(f_i)(x)), \\ &= \bigsqcup T(f_i)(x). \end{aligned}$$

On peut passer de la première ligne à la deuxième parce que les fonctionnelles $T_1, \ldots, T_n$ sont continues. On passe de la deuxième ligne à la troisième en appliquant la définition de la relation d'ordre $\sqsubseteq$ sur un ensemble de fonctions. Pour passer de la troisième ligne à la quatrième, il est nécessaire que chaque fonction f_i soit croissante, ou continue (cela revient au même). On conclut en utilisant la définition de la fonctionnelle T. Le cas de la fonctionnelle S est analogue.

Il reste à considérer le cas de l'alternative. Quelles que soient la chaîne de fonctions croissantes f_i, la fonctionnelle continue B et la valeur de x, les éléments $B(f_i)(x)$ forment une chaîne du domaine primitif étendu $\{true, false, \bot\}$, dont la meilleure borne supérieure est $B(f)(x)$, avec $f = \bigsqcup f_i$. Selon la valeur de cette borne, on a $U(f)(x) = \bot$, $T_1(f)(x)$ ou $T_2(f)(x)$. Dans tous les cas, on a $\bigsqcup U(f_i)(x) = U(f)(x)$; ceci achève la démonstration.

Dans le cas de la fonctionnelle T_1 définissant la factorielle, il était facile de donner la forme explicite des approximations successives $T_1{\uparrow}n$ et de la "limite" $T_1{\uparrow}\omega$.[16] Considérons maintenant le cas moins évident de la fonctionnelle T_5 (Manna 1974), définie sur $[\mathbf{Z}^+ \to \mathbf{Z}^+]$ par l'égalité

$$T_5(f)(x) = \text{if } x > 100 \text{ then } x - 10 \text{ else } f(f(x + 11)).$$

On vérifie facilement que la fonction $f91$ définie par l'égalité

$$f91(x) = \text{if } x > 100 \text{ then } x - 10 \text{ else } 91$$

est un point fixe de T_5. On verra plus loin que ce point fixe est unique. Par contre, la fonctionnelle T_6, définie sur $[(\mathbf{Z}^+ \times \mathbf{Z}^+) \to \mathbf{Z}^+]$ par

$$T_6(f)(x, y) = \text{if } x = y \text{ then } y + 1 \text{ else } f(x, f(x - 1, y + 1)),$$

admet comme points fixes les fonctions f_1, f_2 et f_3 définies par

$$\begin{aligned} f_1(x, y) &= \text{if } x = y \text{ then } y + 1 \text{ else } x + 1, \\ f_2(x, y) &= \text{if } x \geq y \text{ then } x + 1 \text{ else } y - 1, \\ f_3(x, y) &= \text{if } x \geq y \wedge even(x - y) \text{ then } x + 1 \text{ else } \bot. \end{aligned}$$

(La formule $even(x - y)$ est vraie si et seulement si $x - y$ est pair.)

On vérifie facilement que $f_3 \sqsubseteq f_1$ et $f_3 \sqsubseteq f_2$, mais cela ne suffit pas à prouver que f_3 est le plus petit point fixe. Observons aussi que $f_1' : (x, y) \to x + 1$ n'est pas un point fixe (malgré la ressemblance avec f_1), parce que $f_1'(x, \bot) = x + 1$, tandis que $T_6(f_1')(x, \bot) = \bot$. En effet, la condition $x = y$ vaut $\bot$ dès que x ou y vaut $\bot$.

[16] Rappelons que $T_1{\uparrow}n = T_1^n(\emptyset)$ et que $T_1{\uparrow}\omega = \bigsqcup_n T_1^n(\emptyset)$.

Remarque. Plus généralement, il importe de bien voir la différence existant entre l'égalité mathématique usuelle, pour laquelle les valeurs de $3 = \perp$ et $\perp = \perp$ sont respectivement *false* et *true*, et "l'égalité programmée", pour laquelle la valeur commune de ces expressions est $\perp$. L'égalité mathématique n'est ni stricte ni continue, au contraire de l'égalité programmée.

5.3.4. *Evaluation de programmes fonctionnels*

Le paragraphe précédent a montré comment une fonctionnelle continue définit sans ambiguïté une fonction (le plus petit point fixe de la fonctionnelle). La théorie du point fixe fournit aussi un moyen simple pour évaluer une forme, c'est-à-dire la valeur qu'une fonction attribue à des valeurs licites de ses arguments. Soient une fonction f définie par l'équation $f = T(f)$ et une valeur $x \in dom(f)$;[17] la restriction de continuité ne garantit pas la stationnarité de la chaîne fonctionnelle $\{T{\uparrow}n\}$, mais la chaîne non fonctionnelle $\{T{\uparrow}n(x)\}$ reste stationnaire pour toute valeur de x.

La règle d'évaluation des formes consistera donc à engendrer la suite $(T{\uparrow}n(x))$, jusqu'à ce que la limite soit atteinte; le problème est précisément de déterminer quand cette limite est atteinte. Etant donné que le codomaine $(E, \sqsubseteq)$ de la fonction f définie par T est un ensemble non fonctionnel, on distingue deux cas. Si $f(x)$ est entièrement défini (élément maximal de $(E, \sqsubseteq)$), on arrêtera la génération de la suite $(T{\uparrow}n(x))$ dès qu'un élément entièrement défini est produit; dans le cas contraire, la génération ne s'arrêtera pas (faute de critère d'arrêt approprié) et aucune valeur ne sera fournie.

Considérons à nouveau le cas de la factorielle, définie par la fonctionnelle T_1 donnée plus haut. Le calcul de la factorielle de 5 donnera lieu à la suite

$$
\begin{aligned}
(T_1{\uparrow}0)(5) &= \perp, \\
(T_1{\uparrow}1)(5) &= \text{if } 5{=}0 \text{ then } 1 \text{ else } \perp = \perp, \\
(T_1{\uparrow}2)(5) &= 5 * (\text{if } 4{=}0 \text{ then } 1 \text{ else } \perp) = 5 * \perp = \perp, \\
(T_1{\uparrow}3)(5) &= \cdots = 5 * 4 * \perp = \perp, \\
&\ \ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
(T_1{\uparrow}6)(5) &= \cdots = 5 * 4 * 3 * 2 * 1 * 1 = 120.
\end{aligned}
$$

La valeur 120 étant un élément maximal de l'ensemble $\mathbf{N}^+$, le calcul s'arrête.

Ce type de règle d'évaluation suggère que le calcul de $f(x)$ ne sera effectif que si le résultat est un élément maximal de l'ensemble étendu dans lequel f est censé prendre ses valeurs. En général, cela signifiera simplement que $f(x)$ doit être "défini", c'est-à-dire distinct de $\perp$.

---

[17] T est une fonctionnelle définie sur un domaine approprié, dont les éléments sont des fonctions continues. On convient que la même lettre (ici f) sert à poser l'équation de point fixe relative à une fonctionnelle T et à désigner le plus petit point fixe correspondant, soit $T{\uparrow}\omega$.

Ceci suggère une première règle de programmation fonctionnelle. Le calcul de la valeur d'une fonction f ne doit être demandé que pour des valeurs x dont l'utilisateur sait à l'avance que $f(x)$ est défini. L'idéal est de disposer d'un critère simple et général pour reconnaître les valeurs licites, mais ce n'est pas toujours possible; nous reviendrons sur ce point au paragraphe suivant.

La règle d'évaluation liée à la théorie du point fixe (en abrégé, règle du point fixe) pose un autre problème que nous allons maintenant introduire. La fonctionnelle T_7 est définie sur le domaine $[\mathbf{N}^+ \to \mathbf{N}^+]$ par l'égalité

$$T_7(f)(n) \;=\; \text{if } n=0 \vee n=1 \text{ then } n \text{ else } f(n-1)+f(n-2);$$

elle définit la *suite de Fibonacci.* Un calcul typique est

$$\begin{aligned}
&f(4)\,,\\
&f(3)+f(2)\,,\\
&(f(2)+f(1))+(f(1)+f(0))\,,\\
&((f(1)+f(0))+1)+(1+0)\,,\\
&((1+0)+1)+(1+0)\,,\\
&3\,.
\end{aligned}$$

La deuxième approximation est représentée par l'expression $f(3)+f(2)$, comportant deux occurrences de f, qui doivent être toutes deux "expansées" pour obtenir la troisième approximation. On vérifie facilement que la $(n{+}1)$-ième approximation correspondant au calcul de $f(2n)$ contiendra 2^n occurrences de f et sera donc une expression d'une longueur inacceptable dès que n devient relativement grand. On peut remédier à ce problème[18] de deux manières. La plus radicale est de renoncer à l'usage de fonctionnelles qui, comme T_7, conduisent à des approximations fort longues. C'est là un problème relatif à la *conception* des programmes plutôt qu'aux mécanismes de programmation. Nous reviendrons brièvement sur ce point au paragraphe suivant. L'autre remède consiste à n'autoriser qu'un nombre limité d'expansions (le plus souvent, une seule), lors du passage d'une approximation à la suivante. Si par exemple seule l'expansion de l'occurrence la plus à gauche du foncteur f est réalisée à chaque étape, on obtiendra un calcul composé d'un plus grand nombre d'étapes, mais chaque étape sera plus simple. Le temps de calcul ne sera pas significativement modifié, mais la taille de la mémoire nécessaire à ce calcul au moyen d'un ordinateur sera acceptable (d'ordre linéaire en n). L'évaluation de $f(3)$ devient

$$\begin{aligned}
&f(4)\,,\\
&f(3)+f(2)\,,\\
&(f(2)+f(1))+f(2)\,,\\
&((f(1)+f(0))+f(1))+f(2)\,,\\
&((1+f(0))+f(1))+f(2)\,,
\end{aligned}$$

[18] On observe immédiatement que ce problème n'apparaissait pas pour la fonctionnelle T_1; dans ce cas, il n'existait qu'une seule règle de calcul, puisque chaque expression intermédiaire comportait une seule occurrence de f.

$$
\begin{array}{l}
((1+0)+f(1))+f(2)\,,\\
((1+0)+1)+f(2)\,,\\
((1+0)+1)+(f(1)+f(0))\,,\\
((1+0)+1)+(1+f(0))\,,\\
((1+0)+1)+(1+0)\,,\\
3\,.
\end{array}
$$

Malheureusement, il n'est pas évident que le résultat fourni par cette règle soit toujours le bon, c'est-à-dire celui fourni par la règle du point fixe. D'autre part, si on accepte des règles alternatives, la règle "à gauche d'abord" (en anglais, *leftmost first*) n'est pas nécessairement le meilleur choix. Le cas de la fonctionnelle T_5, définie par l'égalité

$$T_5(f)(x) \quad = \quad \text{if } x > 100 \text{ then } x - 10 \text{ else } f(f(x+11))\,,$$

est éclairant à cet égard. Le calcul de $f(98)$, par la règle "à gauche d'abord", commence comme suit :

$$
\begin{array}{l}
f(98)\,,\\
f(f(109))\,,\\
\text{if } f(109) > 100 \text{ then } f(109) - 10 \text{ else } f(f(f(109)+11))\,,\\
\vdots
\end{array}
$$

Le problème de la longueur des expressions reste critique; il l'est moins si on adopte la règle "à gauche et intérieur d'abord" (en anglais, *leftmost-innermost first*). On sélectionne pour expansion l'occurrence la plus à gauche de f dont aucun argument ne comporte d'occurrence de f. Dans le cas présent, on aura

$$
\begin{array}{l}
f(98)\,,\\
f(f(109))\,,\\
f(109-10) = f(99)\,,\\
f(f(110))\,,\\
f(110-10) = f(100)\,,\\
f(f(111))\,,\\
f(111-10) = f(101)\,,\\
101 - 10 = 91\,.
\end{array}
$$

Ces exemples quelque peu simplistes suffisent à poser le problème. En toute généralité, une *règle de calcul* est un algorithme qui, étant donné

1. une expression bien formée e comportant des occurrences d'un foncteur distingué f de type τ et
2. une fonctionnelle (continue) T de type $[\tau \to \tau]$,

fournit une expression bien formée e', obtenue par remplacement dans e d'une ou plusieurs occurrence(s) de f par $T(f)$, éventuellement suivi de l'évaluation de sous-expressions ne comportant pas d'occurrence de f.

Puisque toute expression comportant une occurrence de f reste non définie, et que la règle du point fixe provoque au moins autant d'expansions du foncteur f que toute autre règle de calcul, on a le théorème suivant.

Théorème 5.23. Soit T une fonctionnelle continue et $\mathcal{R}$ une règle de calcul. Si on désigne par $T\uparrow(\mathcal{R}, n)$ la n-ième approximation engendrée par la règle $\mathcal{R}$, on a

$$T\uparrow(\mathcal{R}, n) \sqsubseteq T\uparrow n\,,$$

pour tout $n \in \mathbf{N}$ et, en particulier,

$$\bigsqcup_n T\uparrow(\mathcal{R}, n) \sqsubseteq T\uparrow\omega\,.$$

Autrement dit, la fonction calculée par la règle $\mathcal{R}$ est moins définie ou égale à la fonction calculée par la règle du point fixe. Si en utilisant une règle différente de celle du point fixe, on obtient pour $\bigsqcup_n T\uparrow(\mathcal{R}, n)\,(x)$ une valeur bien définie, on sait que la règle du point fixe conduirait à la même valeur. Si par contre la règle $\mathcal{R}$ conduit à la valeur $\perp$ (en pratique, ne donne pas de valeur), on ne peut pas conclure. Une analyse plus fine des règles de calcul utilisées en pratique dépasserait le cadre d'un chapitre introductif.

En programmation classique, l'évaluation d'une forme $f(x)$ passe par la conception d'un algorithme détaillant le calcul de $f(x)$ en fonction de x.
En programmation fonctionnelle, cette évaluation nécessite seulement l'écriture, dans un formalisme adéquat, d'une "propriété caractéristique" de f. Formellement, cette propriété correspondra à une fonctionnelle continue dont f sera un point fixe. Cette exigence formelle n'est pas une restriction gênante en pratique. Les propriétés que l'on associe aux fonctions prennent la plupart du temps la forme d'une fonctionnelle continue, qu'on le veuille ou non. Par contre, il n'est pas rare que la fonction visée soit plus spécifique que la fonction définie (via la règle du point fixe) par les propriétés fournies.

Considérons un cas extrême mais néanmoins éclairant. Comment définir la multiplication dans le cadre fonctionnel ? La première étape consiste à donner ses propriétés caractéristiques. On peut dire, par exemple, que la multiplication admet 0 pour absorbant, admet le neutre 1 et est commutative. La deuxième étape sera de "coder" ces propriétés sous forme d'une fonctionnelle continue, qui pourrait être T_8 :

$$\begin{aligned} T_8(f)(x,y) \;=\; & \text{if } x = 0 \text{ then } 0 \\ & \text{else if } x = 1 \text{ then } y \\ & \text{else } f(y,x)\,. \end{aligned}$$

La troisième étape sera l'utilisation. Une règle de calcul, celle du point fixe par exemple, permettra de calculer

$$\begin{aligned} &f(0,0) = 0\,,\\ &f(5,0) = 0\,,\\ &f(1,3) = 3\,,\\ &\ldots \end{aligned}$$

Par contre, la plupart des calculs se solderont par un échec. On aura notamment

$$f(5,2) = \perp,$$
$$f(2,3) = \perp,$$
$$\ldots$$

Observons néanmoins qu'aucun résultat erroné ne sera produit; simplement, certaines évaluations ne produiront aucun résultat. Le programmeur-utilisateur déduira de ces échecs que les propriétés fournies ne sont pas suffisantes. Plus formellement, ces propriétés correspondent à une fonctionnelle dont le plus petit point fixe est strictement moins défini que la fonction visée. La conclusion de ceci est qu'il faut adjoindre au paradigme fonctionnel basé sur la théorie du point fixe une méthode permettant d'évaluer le caractère "suffisant" des propriétés fournies pour la définition (programmation) d'une fonction. C'est l'objet du paragraphe suivant.

5.3.5. *Conception de programmes fonctionnels*

La programmation classique permet d'évaluer une fonction f en un point donné de son domaine à condition d'avoir écrit un algorithme de calcul pour cette fonction. La programmation fonctionnelle permet cette évaluation (moyennant la règle du point fixe) à condition d'avoir explicité sous une forme adéquate une propriété suffisamment précise de la fonction f. Quand une propriété est-elle "suffisamment précise" ?

Pour le déterminer, considérons une fois de plus les fonctionnelles T_1 (factorielle) et T_2 (plus grand commun diviseur). Dans le premier cas, on se rend compte immédiatement que le calcul de $f(17)$ est possible si et seulement si celui de $f(16)$ est possible, et ainsi de suite. Dans le second cas, on voit que le calcul de $g(12,16)$ sera possible si et seulement si celui de $g(12,4)$ est possible; ce dernier est lui-même possible si le calcul de $g(8,4)$ est possible, et ainsi de suite.

Les propriétés exprimées par les fonctionnelles T_1 et T_2 sont suffisamment précises parce qu'elles réduisent le calcul pour des données relativement compliquées à un calcul pour des données plus simples (ou, plus généralement, à un nombre fini de tels calculs). Ce n'était pas le cas pour la fonctionnelle T_8. Celle-ci réduisait bien $f(5,0)$ à $f(0,5)$ puis à 0, mais ne réduisait pas effectivement le terme $f(2,3)$, transformé successivement en $f(3,2)$, $f(2,3)$, $f(3,2)$, $f(2,3)$, etc.

Une propriété d'une fonction f permettra la programmation de celle-ci si et seulement si la *relation de réduction* (notée $\preceq$) qu'elle induit sur le domaine de f est un ordre bien fondé (toute suite strictement décroissante est finie). Les éléments minimaux sont les éléments du domaine pour lesquels le calcul est immédiat.

Dans le cas de la factorielle, les relations $\prec$ et $\preceq$ s'identifient aux relations numériques habituelles $<$ et $\leq$ sur $\mathbf{N}$. Dans le cas du plus grand commun diviseur, la description précise de la relation induite par T_2 est incommode

mais on observe que, d'une part,

$$(x,y) \prec (x',y') \;\Rightarrow\; x+y < x'+y'$$

et, d'autre part, que la relation $\rho = \{((x,y),(x',y')) : x+y < x'+y'\}$ est un ordre strict bien fondé sur le domaine de la fonction visée. On remarque aussi que, la relation de réduction induite par T_2 étant *incluse* dans une relation d'ordre bien fondée, aucune "régression infinie" ne sera possible.

Nous avons signalé au paragraphe précédent que la fonctionnelle T_5, définie par l'égalité

$$T_5(f)(x) \;=\; \text{if } x > 100 \text{ then } x-10 \text{ else } f(f(x+11))\,,$$

admettait comme plus petit et unique point fixe la fonction $f91$ définie par l'égalité

$$f91(x) \;=\; \text{if } x > 100 \text{ then } x-10 \text{ else } 91\,.$$

L'exemple du calcul de $f(98)$ suggère une preuve. On voit que ce calcul requiert celui de $f(99)$, celui de $f(100)$ et celui de $f(101)$. Ce dernier est immédiat, comme celui de $f(x)$ pour tout $x > 100$. Ceci suggère la définition suivante :

$$(\forall\, x,y \in \mathbf{Z})\,(x \prec y \;=_{def}\; y < x \leq 101)\,.$$

La relation $\preceq$ est un ordre bien fondé sur $\mathbf{Z}$ et contient la relation de réduction induite par application de la règle "à gauche intérieur d'abord" à la fonctionnelle T_5. Elle permet de démontrer par $\preceq$-induction sur x que $T_5{\uparrow}\omega(x) = f91(x)$, pour tout $x \in \mathbf{Z}$. La démonstration est plus simple que la détermination explicite de la chaîne $\{T_5{\uparrow}n : n \in \mathbf{N}\}$ et est laissée au lecteur.

A titre de comparaison, le lecteur pourra aussi vérifier (par induction naturelle sur $n \in \mathbf{N}$ cette fois), que les approximations successives $T_5 \uparrow n$ de la fonction $f91$ sont données au point x par les expressions suivantes :

- if $x > 100$ then $x-10$ else if $x > 101-n$ then 91 else $\perp$,
  (valable si $n \leq 11$) et
- if $x > 100$ then $x-10$ else if $x > 90-11(n-11)$ then 91 else $\perp$,
  (valable si $n \geq 11$).

d'où on déduit immédiatement que $T_5{\uparrow}\omega(x) = f91(x)$ pour tout $x \in \mathbf{Z}$. Comme la fonction $f91$ est partout définie, et est donc un élément maximal du domaine $[\mathbf{Z}^+ \to \mathbf{Z}^+]$, ce plus petit point fixe est le seul point fixe de T_5.

Remarques. La technique consistant à déterminer explicitement $T{\uparrow}n$ conduit à la détermination exacte de la relation de réduction induite par T; c'est pourquoi elle est généralement d'usage plus délicat que la technique basée sur le choix d'un ordre bien fondé *incluant* cette relation de réduction.[19] Aux chapitres précédents, on avait déjà noté qu'il était plus simple de vérifier ou documenter un programme impératif par un invariant (surensemble d'une trajectoire) que

[19] Le lecteur pourra confirmer ce point en essayant de déterminer la relation de réduction induite par la fonctionnelle T_2, ainsi que la chaîne des approximations successives.

par une trajectoire exacte.
Notons aussi que les deux techniques utilisent l'induction, mais pas de la même manière. La méthode de détermination explicite de la chaîne d'approximations utilise l'induction naturelle, basée sur la relation d'ordre usuelle (et bien fondée) dans $\mathbf{N}$. L'autre méthode utilise une relation d'ordre bien fondée spécifique sur le domaine des données du programme, c'est-à-dire sur le domaine du plus petit point fixe de la fonctionnelle associée au programme. Le domaine étant composé d'objets appelés en informatique *structures de données*, l'induction sur ce domaine se nomme *induction structurelle.*
Enfin, l'induction structurelle peut fournir des renseignements sur le plus petit point fixe d'une fonctionnelle sans que ce dernier soit déterminé explicitement. Considérons par exemple la fonctionnelle T_9, définie par l'égalité

$$\begin{aligned} T_9(f)(x,y) \;=\; & \text{if } x = 0 \text{ then } y^2 + 4y + 11 \\ & \text{else if } y = 0 \text{ then } f(x-1, x^5 .+ f(x-1, x+6) + 7) \\ & \text{else } f(x-1, x^y + f(x, y-1) + 2)\,. \end{aligned}$$

La détermination explicite de $T_9 \uparrow \omega$ semble impossible, mais il est facile de prouver que cette fonction est totale sur $\mathbf{N} \times \mathbf{N}$ et est donc l'unique point fixe de T_9. En effet, l'ordre lexicographique (relation 2 définie ci-dessous) est bien fondé sur le domaine et inclut la relation de réduction induite par T_9.

Pour concevoir un programme fonctionnel, il est commode de choisir explicitement la relation d'ordre bien fondé qui justifiera la "terminaison" du programme, c'est-à-dire le caractère fini du processus d'évaluation basé sur la règle du point fixe. Ce choix n'est évidemment pas unique, et peut influer sur le texte du programme. Considérons d'abord le cas simple de l'addition des naturels, en supposant préexistantes les fonctions successeur et prédécesseur, notées respectivement $add1$ et $sub1$. Le domaine de la fonction d'addition $plus$ est $\mathbf{N} \times \mathbf{N}$; on doit déterminer sur ce domaine une relation d'ordre bien fondé notée $\preceq$ telle que la somme $plus(x,y)$ de deux entiers naturels x et y soit calculable directement (sans recours à la fonction d'addition) ou par l'intermédiaire d'expressions $plus(x',y')$ analogues, mais telles que $(x',y') \prec (x,y)$. Plusieurs choix sont possibles pour cette relation, notamment :

1. $(x',y') \preceq (x,y) =_{def} x' \leq x \wedge y' \leq y\,,$
2. $(x',y') \preceq (x,y) =_{def} x' < x \vee (x' = x \wedge y' \leq y)\,,$
3. $(x',y') \preceq (x,y) =_{def} x' + y' < x + y \vee (x' = x \wedge y' = y)\,.$

De même, plusieurs propriétés de l'addition peuvent servir de base à une définition fonctionnelle, parmi lesquelles

$$\begin{aligned} &\alpha) \quad plus(x,y) \;=\; \text{if } x = 0 \text{ then } y \text{ else } add1(plus(sub1(x), y))\,, \\ &\beta) \quad plus(x,y) \;=\; \text{if } x = 0 \text{ then } y \text{ else } plus(sub1(x), add1(y))\,. \end{aligned}$$

La propriété α convient parce que, si $x \neq 0$, alors $(sub1(x), y) \prec (x,y)$, où $\prec$ est la version stricte d'une quelconque des relations 1, 2 et 3. La propriété β convient parce que, si $x \neq 0$, alors $(sub1(x), add1(y)) \prec (x,y)$, où $\prec$ est la version stricte de la relation 2. Dans les deux cas, les couples de type

$(0, y)$ sont minimaux pour la relation de réduction (quoiqu'ils ne le soient pas nécessairement pour les relations 1, 2 et 3).

De même, pour définir la multiplication, on utilisera la propriété

$$mult(x, y) \quad = \quad \text{if } x = 0 \text{ then } 0 \text{ else } plus(mult(sub1(x), y), y)\,.$$

La programmation fonctionnelle permet donc, dans des cas particuliers mais relativement fréquents, d'interpréter opérationnellement des définitions purement mathématiques (celles de l'addition et de la multiplication avaient déjà été introduites au paragraphe 2.3.3), et donc de les convertir en procédures de calcul.

Nous avons vu au paragraphe 3.3 que les données informatiques ne sont pas toujours atomiques. On doit manipuler non seulement des nombres et autres symboles, mais aussi des assemblages de tels symboles. Un mode d'assemblage particulièrement simple est la *liste*. Une liste est tout simplement une séquence finie. Les listes sont fort semblables aux piles, évoquées au paragraphe 2.3.1. Une liste composée des trois nombres 1, 2 et 3 (dans cet ordre) sera notée $[1, 2, 3]$. La liste vide est notée $[]$. Si une liste L est non vide, son premier élément est noté $car(L)$ et le reste de la liste est noté $cdr(L)$.[20] On a donc $car([1, 2, 3]) = 1$ et $cdr([1, 2, 3]) = [2, 3]$. La fonction *cons*, à deux arguments, réalise l'opération inverse: $cons(1, [2, 3]) = [1, 2, 3]$. A partir de ces fonctions de base, on définit (et donc, on réalise) facilement la fonction de concaténation de deux listes. Cette fonction, notée *append*, est l'unique point fixe de la fonctionnelle associée à la propriété

$$append(X, Y) \quad = \quad \text{if } X = [] \text{ then } Y \text{ else } cons(car(X), append(cdr(X), Y))\,.$$

L'évaluation d'une forme $append(X, Y)$ est immédiate si X est la liste vide. Sinon, elle se réduit à l'évaluation d'une forme de type $append(X', Y')$, où la longueur de X' est la longueur de X moins un. Il est clair que l'évaluation d'une forme se terminera en $n + 1$ étapes, si n est la longueur du premier argument.

5.3.6. *Démonstration automatique*

La logique ou, plus spécifiquement, la théorie du point fixe, permet de fonder avec rigueur la programmation fonctionnelle (et d'autres domaines de l'informatique). Inversement, la programmation fonctionnelle peut contribuer à la logique. De prime abord, le style fonctionnel de définition d'objets mathématiques peut surprendre, notamment à cause de son apparente circularité. Pourtant, ce type de définition existait bien avant l'informatique. En particulier, la définition fonctionnelle (récursive) de l'addition est directement inspirée des axiomes de Peano. La définition récursive de la factorielle est aussi

[20] Ces notations remontent aux origines de la programmation fonctionnelle et sont consacrées par l'usage.

l'une des plus fréquemment utilisées. Le point important est que le style fonctionnel de définition se prête bien à l'étude systématique, voire automatique, des objets définis; en effet, le raisonnement par induction est à la fois très puissant et assez mécanique. Le point délicat est le choix d'une relation d'ordre bien fondé sur le domaine de la fonction dont on étudie les propriétés. Bien que ces relations soient en nombre infini, peu d'entre elles sont utiles en pratique. A titre d'exemple, nous montrons brièvement comment le système de Boyer et Moore (1979) produit automatiquement les preuves des propriétés usuelles de l'addition. La définition récursive de la fonction *plus* a été fournie au système, qui connaît aussi les propriétés des objets de base $0, add1, sub1$. La figure 5.1 comporte une version simplifiée et traduite de ce que produit effectivement le système. Le système décide lui-même de raisonner par induction et de générer des lemmes, qu'il démontre automatiquement.

Le démonstrateur de Boyer et Moore permet d'obtenir les preuves de résultats beaucoup plus profonds, dans des domaines variés. Néanmoins, les difficultés inhérentes à la démonstration automatique sont loin d'être résolues; la recherche sur ce sujet continue activement. Il est intéressant de combiner logique de programmation et démonstration automatique, pour obtenir un vérificateur de programmes. En effet, les informaticiens produisent les programmes beaucoup plus vite que les mathématiciens ne produisent des théorèmes. Malheureusement, il n'y a pas de miracle: la fiabilité des programmes laisse beaucoup à désirer. Les vérificateurs de programmes, de même que les systèmes (semi-)automatiques de programmation, n'ont pas encore atteint un niveau de performance satisfaisant. Des considérations théoriques et pratiques font craindre que la portée de tels systèmes reste fort limitée, du moins dans l'avenir proche.

5.3.7. *Liens avec la programmation séquentielle*

Plutôt que d'étudier un programme fonctionnel directement, il est parfois plus commode de prouver son équivalence avec un programme plus simple. On a par exemple deux fonctionnelles T_{10} et T_{11}, définies par les égalités

$$T_{10}(f)(x,y) \;=\; \text{if } x=0 \text{ then } y \text{ else } f(x-1,y)\circ x\,,$$
$$T_{11}(g)(x,y) \;=\; \text{if } x=0 \text{ then } y \text{ else } g(x-1,y\circ x)\,.$$

Le symbole "$\circ$" représente une opération associative et commutative. On veut prouver que les deux (schémas de) programmes sont équivalents, c'est-à-dire que les deux fonctionnelles ont le même plus petit point fixe. Pour le prouver, on applique le théorème de Scott à la propriété P ci-dessous:

$$P(f,g) =_{def} \forall x,y,r\,[f(x,y)=g(x,y) \wedge f(x,y)\circ r = g(x,y\circ r)]\,.$$

Cette propriété est admissible: $a=b$ s'écrit $a\sqsubseteq b \wedge b\sqsubseteq a$.
Il suffit de prouver $P(T_{10}{\uparrow}\omega, T_{11}{\uparrow}\omega)$ et, d'après le théorème de Scott, ceci se ramène à prouver, d'une part, $P(\emptyset,\emptyset)$, qui se réduit à $\bot = \bot \wedge \bot = \bot$ (la

théorème. $plus(0,x) = x$
preuve. Simple expansion de la définition.
lemme. $plus(x,0) = x$
preuve. La preuve est omise ici.
théorème. $plus(x,y) = plus(y,x)$
preuve. Le système essaie un raisonnement inductif sur la variable x.
Ceci provoque la génération de deux lemmes (cas de base et cas inductif).

lemme 1. $plus(0,y) = plus(y,0)$
preuve du lemme 1.
$plus(0,y) = y$ (expansion)
$plus(y,0) = y$ (lemme précédent)
$plus(0,y) = plus(y,0)$ (propriété de l'égalité)
lemme 2. $x > 0 \wedge plus(sub1(x),y) = plus(y,sub1(x)) \Rightarrow plus(x,y) = plus(y,x)$
preuve du lemme 2.
(on déduit le conséquent de l'antécédent)
$plus(x,y) = add1(plus(sub1(x),y))$ (expansion)
$= add1(plus(y,sub1(x)))$ (antécédent)
(Pour continuer utilement le développement, un lemme est nécessaire)
lemme 3. $add1(plus(u,v)) = plus(u,add1(v))$
(le système raisonne par induction sur u,
ce qui provoque la génération de deux nouveaux lemmes)
lemme 4. $add1(plus(0,v)) = plus(0,add1(v))$
(évident; la valeur commune est $add1(v)$)
lemme 5.
$u > 0 \wedge add1(plus(sub1(u),v)) = plus(sub1(u),add1(v)) \Rightarrow add1(plus(u,v)) = plus(u,add1(v))$
preuve du lemme 5.
$add1(plus(u,v)) = add1(add1(plus(sub1(u),v)))$ (expansion)
$= add1(plus(sub1(u),add1(v)))$ (antécédent)
$= plus(u,add1(v))$ (abstraction)
(la preuve du lemme 3 est achevée)
(la preuve du lemme 2 est reprise)
$= plus(y,add1(sub1(x)))$ (lemme 3)
$= plus(y,x)$ (axiome)
(ceci achève la preuve du théorème)

Figure 5.1 *Démonstrateur de Boyer et Moore : un exemple*

fonction $\circ$ est étendue naturellement), puis à prouver, d'autre part, que pour toutes fonctions f et g on a

$$P(f,g) \Rightarrow P(T_{10}(f),T_{11}(g)).$$

On suppose vrai l'antécédent. Le conséquent s'écrit

$$\forall x, y, r\,[T_{10}(f)(x,y) = T_{11}(g)(x,y) \wedge T_{10}(f)(x,y) \circ r = T_{11}(g)(x, y \circ r)]\,.$$

La première égalité se réduit à $y = y$ ou à $f(x-1, y) \circ x = g(x-1, y \circ x)$ selon que x vaut 0 ou non. Cette égalité est une conséquence de $P(f, g)$.
La seconde égalité du conséquent se réduit à $y \circ r = y \circ r$ si $x = 0$; sinon, elle se démontre comme suit :

$$\begin{array}{lcll} T_{10}(f)(x,y) \circ r & = & [f(x-1,y) \circ x] \circ r & \text{(définition de } f\text{)} \\ & = & f(x-1,y) \circ [x \circ r] & \text{(associativité de } \circ\text{)} \\ & = & g(x-1, y \circ [x \circ r]) & \text{(antécédent)} \\ & = & g(x-1, [y \circ r] \circ x) & \text{(associativité de } \circ\text{)} \\ & = & T_{11}(g)(x, y \circ r) & \text{(définition de } g\text{)} \end{array}$$

Remarque. Pour prouver $f = g$, on a dû énoncer et démontrer une propriété plus forte. Ce phénomène est classique dans les preuves de programmes.

Les deux schémas de programmes associés à T_{10} et T_{11} sont fort semblables mais le second présente une particularité intéressante. L'évaluation de $g(x, y)$ se ramène soit à l'évaluation d'une expression ne comportant pas g, soit à une évaluation du type $g(\alpha, \beta)$, où α et β ne contiennent pas g. On dit que la récursivité est *dégénérée*, parce que l'évaluation de la forme $g(x, y)$ peut se faire simplement par génération des couples successifs (α, β). Cette idée est formalisée par le théorème suivant, qui établit un lien entre programmation fonctionnelle et programmation impérative.

Théorème 5.24. Soient D un domaine non fonctionnel, b un prédicat sur D, g une transformation de D et φ le plus petit point fixe de la fonctionnelle T_{12} de domaine $[D^+ \to D^+]$ définie par l'égalité

$$T_{12}(f)(x) \quad = \quad \text{if } b(x) \text{ then } x \text{ else } f(g(x))\,.$$

Soit A le programme impératif

$$\text{while } \neg b(x) \text{ do } x := g(x)\,.$$

Le triplet de Hoare $\{x = x_0\}\, A\, \{x = \varphi(x_0)\}$ est vrai pour toute valeur $x \in D$.
Démonstration. Soit $x_0 \in D$. On considère la suite $(x_0, g(x_0), g(g(x_0)), \ldots)$. Si pour tout élément u de cette suite $b(u)$ est faux, alors la valeur de $\varphi(x_0)$ est $\perp$ et l'exécution de la boucle A d'état initial $x = x_0$ est infinie. Sinon, soit k le plus petit entier tel que $b(g^k(x_0))$ soit vrai. On a alors $\varphi(x_0) = g^k(x_0)$. L'exécution de A dont l'état initial est $x = x_0$ comporte alors $k+1$ états dont le dernier est $x = \varphi(x_0)$.

Ce théorème suggère que la programmation séquentielle peut être vue comme un sous-ensemble de la programmation fonctionnelle (on se restreint à la récursivité dégénérée, c'est-à-dire à l'itération). Cette vue n'est pas fausse, mais il ne faut pas en déduire que la programmation fonctionnelle est plus riche que la programmation séquentielle. Pour réaliser dans le paradigme séquentiel l'évaluation d'une forme, c'est-à-dire d'un programme fonctionnel appliqué à des arguments adéquats, il suffit d'utiliser la règle du point fixe, qui est de

nature séquentielle. Le programme séquentiel utilisera comme données lors d'un calcul $f(x)$ le texte de la fonctionnelle T dont f est le plus petit point fixe, ainsi que l'argument x. Il se composera d'une simple boucle et générera la suite des valeurs $T\uparrow n(x)$ pour n croissant, jusqu'à l'obtention d'une valeur définie, distincte de $\perp$.

Sans entrer dans les détails, signalons que dans les systèmes informatiques actuels les formes sont évaluées séquentiellement. Les techniques d'évaluation utilisées sont plus sophistiquées et plus efficaces que celle esquissée ici, mais le principe est le même.

5.4. Application à la programmation impérative

Il n'est pas indispensable de recourir à la théorie du point fixe pour étudier les programmes impératifs; la logique de Hoare, directement basée sur la théorie des ensembles, est suffisante. Néanmoins, la théorie du point fixe permet de mieux comprendre la notion d'itération, et aussi de justifier des extensions couramment introduites en programmation séquentielle structurée. Rappelons qu'un système formel séquentiel ou parallèle est équivalent à une itération non déterministe. Les résultats donnés ici pour la programmation séquentielle structurée s'appliquent aussi à la programmation non structurée, avec ou sans parallélisme.

5.4.1. *La sémantique d'une boucle*

Considérons un programme S dont la sémantique relationnelle est $R(S)$. On se donne aussi une condition B, de sémantique $R(B)$. Rappelons que si toutes les variables évoquées par S et/ou B appartiennent à un ensemble V, on peut assimiler $R(S)$ et $R(B)$ à des relations binaires sur Σ_V, c'est-à-dire à des sous-ensembles de $\Sigma_V \times \Sigma_V$. Dans ce contexte, Σ_V est l'ensemble des applications σ de domaine V qui, à chaque variable $v \in V$ associent une valeur licite pour v. Cette valeur est notée $val_\sigma(v)$, $\sigma(v)$, σ_v ou encore v_σ; ces notations sont utilisées non seulement pour une simple variable $v \in V$, mais aussi pour une expression évoquant seulement les variables de V.

On a vu au paragraphe 3.5.3.2 que la sémantique du programme

$$K \ : \ \text{while } B \text{ do } S$$

était la relation

$$R(K) \ = \ \bigcup_{n \in \mathbf{N}} R_n(K)\,,$$

où la suite $(R_n(K) : n \in \mathbf{N})$ est définie par les égalités

$$R_0(K) \ = \ R(\neg B)\,,$$
$$R_{n+1}(K) \ = \ (R_n(K) \circ R(S) \circ R(B)) \cup R(\neg B)\,.$$

Ce résultat se retrouve facilement par la théorie du point fixe. L'ensemble $\mathcal{P}(\Sigma_V \times \Sigma_V)$ dont les éléments sont les relations binaires sur Σ_V, muni de la relation d'inclusion, est un ensemble ordonné inductif. Le minimum de cet ensemble est l'ensemble vide, et la meilleure borne supérieure $\bigsqcup \mathcal{C}$ d'une chaîne $\mathcal{C}$ est la réunion $\bigcup \mathcal{C}$.

Etant donné deux relations $U, V \subset \Sigma_V \times \Sigma_V$, la fonctionnelle $\mathcal{T}$ définie par l'égalité

$$\mathcal{T}(X) \;=\; (X \circ U) \cup V$$

est visiblement croissante,

$$X \subset X' \;\Rightarrow\; \mathcal{T}(X) \subset \mathcal{T}(X')\,,$$

et continue,

$$\mathcal{T}(\bigcup_{i\in I} X_i) \;=\; \bigcup_{i\in I} \mathcal{T}(X_i)\,.$$

(Il n'est même pas nécessaire que $\{X_i : i \in I\}$ soit une chaîne; ce peut être un sous-ensemble quelconque de $\mathcal{P}(\Sigma_V \times \Sigma_V)$.)

Le plus petit point fixe de la fonctionnelle $\mathcal{T}$ est donc

$$\mathcal{T}{\uparrow}\omega = \bigcup_{n\in\mathbf{N}} \mathcal{T}{\uparrow}n\,;$$

on a aussi

$$\mathcal{T}{\uparrow}0 \;=\; \emptyset\,,$$
$$\mathcal{T}{\uparrow}(n{+}1)(X) \;=\; (\mathcal{T}{\uparrow}n(X) \circ U) \cup V\,.$$

Avec les notations utilisées au paragraphe précédent, la forme classique de l'approximation de la sémantique $R(K)$ de la boucle K est la relation

$$R_n(K) = \mathcal{T}{\uparrow}(n{+}1)\,,$$

où on a fixé les relations U et V par les égalités $U = R(S) \circ R(B)$ et $V = R(\neg B)$.

5.4.2. *Les invariants d'une boucle*

La règle de Hoare relative à la boucle est

$$\frac{\{I \wedge B\}\ S\ \{I\}}{\{I\}\ \text{while } B \text{ do } S\ \{I \wedge \neg B\}}$$

De prime abord, cette règle n'est pas différente des autres. Nous avons cependant observé que la découverte de l'invariant I était généralement le seul point délicat lors de la conception ou de l'analyse d'une boucle. Par exemple, pour démontrer le triplet de Hoare

$$\{x{=}x_0 \wedge y{=}1\}\ \text{while } x > 0 \text{ do } (y,x) := (y * x, x-1)\ \{y{=}x_0!\}\,,$$

nous avons dû considérer la formule

$$I \;=_{def}\; [0 \le x \le x_0 \wedge y * x! = x_0!]\,,$$

plus complexe que la précondition et la postcondition du triplet à démontrer.

Dans ce paragraphe, nous utilisons la théorie du point fixe pour caractériser les invariants susceptibles de valider une boucle. On considère donc la boucle

$$K =_{def} \text{ while } B \text{ do } S,$$

et on suppose que l'exécution de S se termine toujours, pour peu que l'état initial satisfasse la condition B; cela permet d'assimiler $B \longrightarrow S$ à une affectation conditionnelle, sur le plan sémantique du moins. Pour prouver le triplet

$$\{P\}\ K\ \{Q\} \tag{5.1}$$

il faut, d'après la logique de Hoare, trouver un invariant I solution du système

$$\begin{array}{ll} & P \Rightarrow I, \\ (5.2) & \{I\}\ (B \longrightarrow S)\ \{I\}, \\ & (I \wedge \neg B) \Rightarrow Q. \end{array}$$

Le triplet 5.1 et le système 5.2 donnent lieu à plusieurs problèmes, selon la répartition des objets P, I, B, S et Q en objets connus et objets inconnus. Nous allons considérer certains de ces problèmes (pour plus de détails, voir notamment Clarke 1977 et 1980, Cousot 1977, 1978 et 1989, Lamport 1987 et Van Lamsweerde et Sintzoff 1979).

5.4.2.1. *Problème canonique direct*

Si la précondition P et la boucle K (c'est-à-dire, la condition B et l'affectation S) sont fixées, le problème canonique direct consiste à déterminer I et Q.[21] Ce problème admet visiblement la solution triviale $I = Q = \mathit{true}$. Les solutions les plus intéressantes seront ici les solutions les plus restrictives, ou les plus fortes, c'est-à-dire les plus "petites" au sens de l'ordre induit par l'implication. On sait que si Q est une solution du triplet 5.1, toute formule Q' impliquée par Q est aussi une solution. On cherchera donc la solution la plus forte, que par abus d'écriture on peut noter $slp[P; K]$. (C'est un abus d'écriture parce que K n'est pas une affectation.) La signification intuitive de cette formule est claire : si la boucle K est exécutée au départ d'un état vérifiant la précondition P, l'assertion $slp[P; K]$ est la caractérisation la plus précise que l'on puisse donner de l'état final de l'exécution.

Le lien entre la formule $slp[P; K]$ et l'invariant I est fourni par la troisième formule du système 5.2, où l'implication devient une égalité;[22] la solution Q du

[21] On s'intéresse surtout à l'assertion Q, mais la validation du triplet 5.1 ne pourra se faire que grâce à l'invariant I.

[22] Les assertions sont assimilées à des ensembles d'états et nous utiliserons donc le symbole d'égalité plutôt que celui d'équivalence. Pour rappeler la nature logique des assertions, nous conserverons néanmoins l'usage des symboles logiques $\wedge, \vee, \Rightarrow, \mathit{false}$ au lieu des symboles ensemblistes $\cap, \cup, \subset, \emptyset$.

triplet 5.1 sera $(I \wedge \neg B)$, où I est le plus fort invariant de la boucle K, impliqué par la précondition P. On utilise la notation $sin[P;(B \longrightarrow S)]$ pour désigner cet invariant. Son statut est réglé par un théorème.

Théorème 5.25. Le plus fort invariant $sin[P;(B \longrightarrow S)]$ de la boucle $K = \text{while } B \text{ do } S$, impliqué par la condition initiale P, est le plus petit point fixe de la fonctionnelle continue

$$\mathcal{S} : \mathcal{P}(\Sigma_V) \rightarrow \mathcal{P}(\Sigma_V)$$

définie par l'égalité

$$\mathcal{S}(J) = P \vee slp[J;(B \longrightarrow S)] .$$

Démonstration. L'ensemble $(\mathcal{P}(\Sigma_V), \Rightarrow)$ est ordonné inductif (c'est même un treillis complet). Le système

$$P \Rightarrow I ,$$
$$\{I\}\ (B \longrightarrow S)\ \{I\}$$

peut se récrire

$$P \Rightarrow I ,$$
$$slp[I;(B \longrightarrow S)] \Rightarrow I ,$$

ou encore

$$(P \vee slp[I;(B \longrightarrow S)]) \Rightarrow I .$$

La théorie du point fixe s'applique ici. Les solutions de l'équation en I associée à l'inéquation ci-dessus forment un ensemble dont la meilleure borne inférieure est le plus petit point fixe de la fonctionnelle $\mathcal{S}$.

Remarque. Quelle que soit l'affectation conditionnelle $(B \longrightarrow A)$, les transformations $S : J \rightarrow slp[J;(B \longrightarrow A)]$ et $W : J \rightarrow wlp[(B \longrightarrow A);J]$ sont respectivement $\vee$-*additive* et $\wedge$-*additive*, ce qui signifie que pour toute partie $\mathcal{J} = \{J_r : r \in R\}$ de $\mathcal{P}(\Sigma)$ on a

$$S(\bigvee_r J_r) = \bigvee_r S(J_r) ,$$
$$W(\bigwedge_r J_r) = \bigwedge_r W(J_r) .$$

Cela implique la continuité des transformations S et $\mathcal{S}$, pour la relation d'ordre induite par l'implication, et la continuité des transformations duales W et $\mathcal{W}$, pour la relation d'ordre inverse. La transformation $\mathcal{W}$ sera utilisée plus loin; elle est définie par l'égalité

$$\mathcal{W}(J) = Q \wedge wlp[(B \longrightarrow S);J] .$$

Corollaire 5.26.

$$sin[P;(B \rightarrow S)] = \mathcal{S}\uparrow\omega = \bigcup_{n \in \mathbf{N}} slp^n[P;(B \longrightarrow S)] ,$$

où on pose $slp^0[P;X] = P$ et $slp^{n+1}[P;X] = slp[slp^n[P;X];X]$.

Démonstration. On vérifie facilement par récurrence sur n que

$$\bigcup_{i=0}^{n} slp^i[P;(B \longrightarrow S)] = \mathcal{S}\uparrow(n+1)\,.$$

Remarque. Intuitivement, $slp^n[P;(B \longrightarrow S)]$ comporte les états atteignables à partir d'un état initial vérifiant P en exécutant n fois le corps de la boucle K.

A titre d'application, calculons l'expression $slp[P;K]$ pour

$$\begin{aligned}
P &=_{def} y = 1 \wedge x = x_0 \in \mathbf{Z}\,,\\
K &=_{def} \text{while } x \neq 0 \text{ do } (y,x) := (y * x, x-1)\,,\\
B &=_{def} x \neq 0\,,\\
S &=_{def} (y,x) := (y * x, x-1)\,.
\end{aligned}$$

On obtient successivement

$$\begin{aligned}
slp^0[P;(B \longrightarrow S)] &= (x = x_0 \wedge x_0 \in \mathbf{Z})\,,\\
slp^1[P;(B \longrightarrow S)] &= (x = x_0 - 1 \wedge x_0 \in \mathbf{Z}\backslash\{0\} \wedge y = x_0)\,,\\
slp^2[P;(B \longrightarrow S)] &= (x = x_0 - 2 \wedge x_0 \in \mathbf{Z}\backslash\{0,1\} \wedge y = x_0 * (x_0 - 1))\,,\\
&\vdots
\end{aligned}$$

On vérifie facilement par récurrence que $slp^n[P;(B \longrightarrow S)]$ est

$$\begin{aligned}
& x = x_0 - n\\
\wedge\ & x_0 \in \mathbf{Z}\backslash\{0,1,\ldots,n-1\}\\
\wedge\ & y = x_0 * (x_0 - 1) * \cdots * (x_0 - n)\,.
\end{aligned}$$

En utilisant le corollaire 5.26, on obtient

$$\begin{aligned}
sin[P;(B \to S)] &= (0 \leq x \leq x_0 \vee x \leq x_0 < 0)\\
&\quad \wedge\ y = x_0 * (x_0 - 1) * \cdots * (x + 1)\,.
\end{aligned}$$

On en déduit enfin

$$\begin{aligned}
slp[P;K] &= (sin[P;(B \to S)] \wedge \neg B)\,,\\
&= (0 \leq x_0 \wedge y = x_0!)\,.
\end{aligned}$$

Remarque. Si on remplace la condition $B : x \neq 0$ par la condition $B' : x > 0$, on obtient une boucle K' pour laquelle

$$\begin{aligned}
sin[P;(B' \to S)] &= (x = x_0) \vee\\
&\quad (0 \leq x \leq x_0 \wedge y = x_0 * (x_0 - 1) * \cdots * (x + 1))\,,
\end{aligned}$$

et

$$\begin{aligned}
slp[P;K'] &= (sin[P;(B' \to S)] \wedge \neg B')\,,\\
&= (x_0 < 0 \wedge y = 1) \vee (0 \leq x_0 \wedge y = x_0!)\,.
\end{aligned}$$

5.4.2.2. *Problème canonique inverse*

Si la boucle K (c'est-à-dire, la condition B et l'affectation S) et la postcondition Q sont fixées, le problème canonique inverse consiste à déterminer

les formules P et I. Les solutions les plus intéressantes seront ici les solutions les moins restrictives, ou les plus faibles, c'est-à-dire les plus "grandes" au sens de l'ordre induit par l'implication. Deux variantes sont possibles, conduisant pour P à deux solutions notées $wlp[K;Q]$ et $wp[K;Q]$.

Par définition, si la boucle K est exécutée au départ d'un état vérifiant $wlp[K;Q]$ et si l'exécution se termine, alors l'état final vérifie Q. Si la boucle K est exécutée au départ d'un état vérifiant $wp[K;Q]$, alors l'exécution se termine et l'état final vérifie Q. (La condition $wp[K;Q]$ est donc plus forte, plus restrictive, que la condition $wlp[K;Q]$.)

L'obtention de ces deux assertions passe par la détermination du plus faible invariant de K impliquant Q. Cet invariant est noté $win[(B \to S);Q]$ et donne lieu au théorème suivant :

Théorème 5.27. Le plus faible invariant $win[(B \to S);Q]$ de la boucle $K =$ while B do S, impliquant la condition finale Q, est le plus grand point fixe de la fonctionnelle inversement continue

$$\mathcal{W} : \mathcal{P}(\Sigma_V) \longrightarrow \mathcal{P}(\Sigma_V)$$

définie par l'égalité

$$\mathcal{W}(J) = Q \wedge wlp[(B \longrightarrow S);J] .$$

("Inversement continue" signifie "continue pour l'ordre inverse".)
La démonstration se déduit de celle du théorème 5.25, par dualité.

Corollaire 5.28.

$$win[(B \to S);Q] = \mathcal{W}{\downarrow}\omega = \bigcap_{n\in\mathbf{N}} wlp^n[(B \longrightarrow S);Q] ,$$

où on pose $wlp^0[X;Q] = Q$ et $wlp^{n+1}[X;Q] = wlp[X;wlp^n[X;Q]]$.
La démonstration se déduit de celle du corollaire 5.26, par dualité.
Remarque. Intuitivement, $wlp^n[(B \longrightarrow S);Q]$ comporte les états à partir desquels on ne peut atteindre, en exécutant n fois le corps de la boucle K, qu'un état vérifiant Q.

La théorie des invariants que nous venons d'esquisser s'applique aussi aux systèmes non déterministes, non structurés et parallèles. Revenons par exemple au système $\mathcal{S}$ représenté à la figure 3.6, qui permet de calculer le plus grand commun diviseur de trois entiers plus grands que 0. L'invariant que nous avons utilisé au paragraphe 3.4.5 pour valider ce système était le plus faible invariant du système par rapport à la postcondition requise. Plus précisément, si on suppose que le type des trois variables x, y et z est $\mathbf{N}$ (et non $\mathbf{Z}$), on a

$$wp[\mathcal{S};\ x = y = z = n] = (x, y, z > 0 \wedge pgcd(x, y, z) = n) ,$$

si n est strictement positif; par contre

$$wp[\mathcal{S};\ x = y = z = 0] = (x = y = z = 0) .$$

Pour la version libérale de l'opérateur wp, les résultats sont légèrement

différents. On a

$$wlp[\mathcal{S};\ x = y = z = n] \ = \ (pgcd(x, y, z) = n)\,,$$

si n est strictement positif; en effet, si initialement certaines variables sont nulles mais pas toutes, l'exécution ne se terminera pas, ce qui est autorisé ici. On a aussi

$$wlp[\mathcal{S};\ x = y = z = 0] \ = \ (x = 0 \vee y = 0 \vee z = 0)\,.$$

5.4.2.3. *Problème de la vérification*

Si dans le triplet $\{P\}\,K\,\{Q\}$ les trois arguments sont fixés, la seule inconnue est l'invariant I permettant la validation. On a à ce propos le théorème suivant, corollaire des théorèmes précédents.

Théorème 5.29. Etant donné le triplet

$$\{P\} \text{ while } B \text{ do } S\ \{Q\}\,,$$

les cinq conditions suivantes sont équivalentes :

1. Le triplet est valide.
2. La formule $sin[P;\ (B \rightarrow S)] \ \Rightarrow \ win[(B \rightarrow S);\ (B \vee Q)]$ est valide.
3. La formule $sin[P;\ (B \rightarrow S)] \ \Rightarrow \ (B \vee Q)$ est valide.
4. La formule $P \ \Rightarrow \ win[(B \rightarrow S);\ (B \vee Q)]$ est valide.
5. Il existe un invariant I de K vérifiant $(P \ \Rightarrow \ I)$ et $(I \ \Rightarrow \ (B \vee Q))$.

Ce théorème exprime aussi la "complétude de la méthode des invariants" : un triplet relatif à une boucle est valide si et seulement si un invariant permet de le valider.

5.4.2.4. *Problème de la synthèse*

Une version simple de ce problème consiste à chercher une condition B, en fonction de S et I, telle que I soit un invariant de la boucle K; on doit donc avoir

$$\{I \wedge B\}\ S\ \{I\}$$

ou encore

$$(I \wedge B) \ \Rightarrow \ wlp[S; I]\,.$$

La solution la plus intéressante est ici la plus faible, c'est-à-dire

$$B \ =_{def} \ (I \ \Rightarrow \ wlp[S; I])\,.$$

5.5. Domaines sémantiques

La théorie du point fixe a permis de formaliser la sémantique des programmes, c'est-à-dire leur signification. Cette formalisation est précise et commode mais elle est loin d'être complète. Une présentation complète dépasserait le cadre de ce chapitre, et nous n'abordons ce problème que dans la mesure où les débuts d'une de ses solutions les plus élégantes reposent directement sur la théorie du point fixe. Dans cette section, nous montrons d'abord brièvement une carence de la "mini-sémantique" des programmes introduite dans les paragraphes précédents. Cela fait apparaître assez clairement que les "domaines" que nous avons utilisés (par exemple $\mathcal{P}(\Sigma \times \Sigma)$ en programmation impérative et $[D \to D]$ en programmation fonctionnelle) ne sont que des cas particuliers de domaines plus complexes, dont l'usage est indispensable pour étudier les programmes réels. Ces domaines sont souvent définis récursivement, ce qui amène une nouvelle intervention de la théorie du point fixe. (Le lecteur non informaticien trouvera peut-être ce paragraphe un peu aride; il peut passer directement au paragraphe 5.6.)

5.5.1. *Vers une sémantique des programmes fonctionnels*

En programmation fonctionnelle, nous avons pu attribuer à chaque programme "simple" un objet mathématique qui traduit sa signification, à savoir le plus petit point fixe de la fonctionnelle associée. Malheureusement, un programme fonctionnel "réel" ne se réduit pas à un ensemble de programmes du type de ceux que nous avons vus. En particulier, la définition d'une fonction complexe nécessite la définition de fonctions auxiliaires, qui peuvent elles-mêmes reposer sur la définition d'autres fonctions. La réalité conceptuelle sera donc un arbre, ou une hiérarchie de fonctions, alors que notre modèle ne rend compte que d'une fonction isolée ou (moyennant quelques adaptations), d'un ensemble non structuré de fonctions. Théoriquement, cette lacune n'est pas rédhibitoire. Si la définition d'une fonction α nécessite les définitions des fonctions β et γ, et si la fonction β nécessite la définition de la fonction δ, il suffira de produire des définitions pour δ, β, γ et enfin α. L'ensemble des définitions sera étiqueté "programme α".

L'absence de modularisation rendrait néanmoins impraticable la définition d'une fonction complexe, nécessitant par exemple la définition de milliers de fonctions auxiliaires. La plupart de ces fonctions n'ont qu'un intérêt momentané et local, comme en mathématique certains lemmes ne sont énoncés et démontrés qu'en vue de la démonstration d'un théorème. De telles fonctions ne devraient pas être "visibles" au niveau supérieur, où sont définies les fonctions principales. Le simple fait de *nommer* des fonctions auxiliaires peut être dangereux; sommes-nous sûrs que la fonction 2496 n'a pas reçu accidentellement le même nom que la fonction 365? Le fait que les gros programmes soient écrits en équipe augmente les risques.

Les langages fonctionnels comportent donc la possibilité d'imbriquer des définitions de fonctions à l'intérieur d'autres définitions. Il est également possible, au moyen de la "notation lambda" d'évoquer et d'utiliser des fonctions sans les nommer. Par exemple, la fonction

$$\lambda\ (x, y)\ (x^2 + y^2)$$

est bien identifiable (elle calcule la somme des carrés de ses deux arguments) mais n'a pas de nom.

Nous avons donc un problème d'échelle. Un modèle satisfaisant pour un ensemble de quelques fonctions cesse de l'être pour des ensembles plus vastes et plus structurés. Ce problème à l'échelle macroscopique se double d'un problème à l'échelle microscopique. Nous n'avons pas donné de règle pour l'interprétation détaillée des constituants d'un programme fonctionnel. Les "mots" d'un tel programme appartiennent à des catégories diverses; "if" est un mot-clef, "x" est souvent une variable numérique, "f" une constante ou une variable fonctionnelle, "3.14" une constante numérique, etc. Les paramètres, en particulier, posent un problème. On peut s'attendre à ce que, dans un gros programme, x et y soient utilisés plusieurs fois pour identifier des variables différentes. Les modèles sémantiques que nous avons utilisés contournent le problème de la correspondance entre les *identificateurs* (noms) et les objets désignés (fonctions, variables, emplacements en mémoire, etc.). Ce problème est pourtant central en pratique. Il doit être résolu pour obtenir une sémantique complète des programmes et des langages de programmation. Une telle sémantique est *dénotationnelle* ou *compositionnelle*, en ce sens que la sémantique attribuée à une entité composée ne dépend que des sémantiques attribuées aux composants, ainsi que du mode de composition, mais pas des composants eux-mêmes. Cette propriété permet la substitution par équivalents; si S et S' ont même sémantique, et si T admet S comme composant, le programme T' obtenu en remplaçant dans T le composant S par S' sera équivalent à T.

5.5.2. *Vers une sémantique des programmes impératifs*

Il semblerait a priori que le modèle introduit au paragraphe 3.5.3 pour la programmation séquentielle structurée soit plus satisfaisant; il présente en effet une certaine modularité. La sémantique relationnelle d'une séquence, par exemple, s'obtient en composant les sémantiques relationnelles des éléments de la séquence; il en va de même pour l'alternative et l'itération. Nous avons aussi introduit un moyen pour "abstraire" un programme en une affectation : c'est le mécanisme des procédures, ou *abstraction procédurale.* Nous avons cependant ignoré à nouveau le problème de la correspondance, et aussi le problème microscopique : comment une affectation est-elle écrite et interprétée ? Nous n'avons donc pas obtenu une sémantique dénotationnelle, qui permettrait d'écrire

$$\mathcal{S}(x := e) := \mathcal{A}(\mathcal{S}(x), \mathcal{S}(e)) .$$

Dans cette écriture, la fonction sémantique $\mathcal{S}$ attribue une sémantique non seulement aux programmes mais à toute entité intervenant comme constituant d'un programme. La fonction $\mathcal{A}$ détermine la sémantique d'une affectation en fonction des sémantiques respectives des deux membres de l'affectation.

Notre objectif ici n'est pas de faire l'inventaire des difficultés posées par la sémantique formelle des programmes et des langages de programmation, et encore moins de résoudre ces difficultés. Nous avons cependant isolé certains points, parce que la théorie du point fixe apporte une contribution majeure à leur solution.

5.5.3. *Exemples de domaines définis récursivement*

Nous avons interprété les programmes comme des objets mathématiques (fonctions pour les programmes fonctionnels, relations pour les programmes impératifs). Ces fonctions, de même que ces relations, forment les domaines sémantiques. Tout ceci peut être conservé mais, pour résoudre les problèmes évoqués, il faudra introduire d'autres domaines sémantiques. Ceux-ci serviront à l'interprétation des objets "auxiliaires" de la programmation. Par exemple, un *environnement* sera une fonction qui à chaque identificateur attribue une valeur. Ces fonctions forment le domaine des environnements. Les valeurs elles-mêmes forment des domaines. La nature de ces domaines n'est pas toujours évidente; c'est ici que la théorie des domaines intervient.

Considérons d'abord un exemple simple de domaine dont la définition est naturellement récursive, celui des "listes d'entiers", que nous appelons D. Nous souhaitons non seulement qu'il comporte des listes simples comme `()` et `(3 -2)`, mais aussi des listes "composites" comme `(() (3))` et `((4 ()) (6) ((5)))`. Nous pouvons noter $\langle A \rangle$ l'ensemble des listes simples constituées d'éléments de A. Une liste de listes simples, telle `(() (3))`, sera élément de l'ensemble $\langle\langle \mathbf{Z} \rangle\rangle$ et une liste "mixte", dont les éléments sont des entiers ou des listes simples d'entiers, appartiendra à l'ensemble $\langle \mathbf{Z} \cup \langle \mathbf{Z} \rangle \rangle$.[23] On devine que l'expression formelle représentant D sera compliquée ... voire infinie, puisqu'aucune borne n'est imposée au niveau d'imbrication des listes. En fait, il est plus simple de "définir" D par une équation telle que

$$D = \langle \mathbf{N} \cup D \rangle . \tag{5.3}$$

Cette équation peut se lire de manière constructive comme suit. Pour obtenir un nouvel élément de D, prendre une suite finie dont les éléments appartiennent à $\mathbf{N}$ ou à D, et l'entourer de parenthèses. Le problème est naturellement de donner une signification précise à la définition de "domaines récursifs" au moyen d'équations de point fixe.

[23] Les parenthèses angulaires dénotent ici un opérateur monadique.

L'exemple du domaine D et, plus généralement, des ensembles de données récursives (listes, arbres) est éclairant mais ne semble pas justifier le recours à une machinerie théorique abstraite. On peut par exemple identifier un élément de D à un arbre fini dont les feuilles sont des entiers ou la liste vide, comme illustré à la figure 5.2. Cette définition peut être formalisée facilement, sans recours à une théorie spécifique.

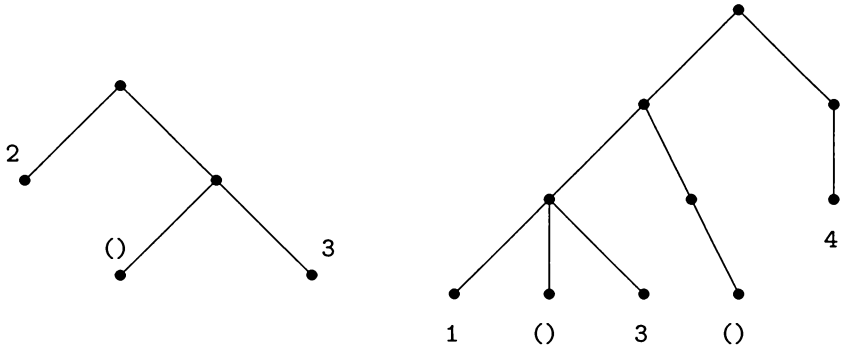

Figure 5.2 *Listes arborescentes* `(2 (() 3))` *et* `(((1 () 3) (())) (4))`

Par contre, les domaines auxquels la sémantique dénotationnelle des langages de programmation donne lieu apparaissent uniquement sous forme implicite, comme solutions d'équations de type $E = \phi(E)$. En programmation fonctionnelle, le domaine E des "valeurs" pourra contenir des éléments des domaines de données habituels (entiers, listes, etc.) mais aussi des fonctions ou des fonctionnelles, dont les arguments et valeurs seront eux-mêmes des éléments de E. On pourrait donc avoir

$$E \;=\; V \cup (E \to E) \cup \cdots$$

Il ne semble plus possible ici de contourner la résolution de l'équation définissant le domaine E, parce que cette équation est la seule information que nous ayons au sujet de E.

5.5.4. *Construction de domaines définis récursivement*

Le lecteur est à présent assez aguerri pour deviner que l'ensemble défini par l'équation 5.3 est $\bigcup_{n\in\mathbf{N}} D_n$, où la suite $(D_n : n \in \mathbf{N})$ est définie par récurrence comme suit :

$$D_0 \;=\; \langle \mathbf{N} \rangle\,,$$
$$D_{n+1} \;=\; \langle \mathbf{N} \cup D_n \rangle\,.$$

On démontre facilement par récurrence que la suite est croissante; on a $D_n \subset D_{n+1}$ pour tout n.

Il faut quand même se méfier de cette construction, parce qu'elle ne conduit pas toujours à une solution de l'équation. Considérons par exemple un domaine E, version simplifiée du domaine sémantique introduit plus haut, et censé regrouper l'ensemble $\mathbf{Z}$, l'ensemble $(\mathbf{Z} \to \mathbf{Z})$ des transformations sur $\mathbf{Z}$, l'ensemble des transformations sur $(\mathbf{Z} \to \mathbf{Z})$, etc. Ce domaine semble être solution de l'équation

$$E = \mathbf{Z} \cup (E \to E) . \tag{5.4}$$

Il se fait pourtant que l'équation (5.4) n'a pas de solution. C'est une conséquence du théorème de Cantor: si le cardinal de l'ensemble E est $\aleph_\alpha$, pour un certain ordinal α, celui de $\mathbf{Z} \cup (E \to E)$ sera $\aleph_\beta$, avec $\beta > \alpha$.[24]

Pour justifier la construction du domaine D des listes d'entiers, il suffit de démontrer la continuité de l'opérateur ϕ intervenant dans l'équation de point fixe relative à D. Cet opérateur est la composition de deux opérateurs plus simples,

$$\alpha : D \longrightarrow A \cup D ,$$
$$\beta : D \longrightarrow \langle D \rangle .$$

Il est commode de traiter ces opérateurs séparément. Pour démontrer leur continuité, il faut assigner à ces opérateurs des domaines de définition ordonnés et inductifs. Cela pose déjà un problème, parce que le domaine de définition idéal serait ... le domaine que l'on cherche à définir! Dans le cas présent, ce problème est contourné aisément. On choisit comme domaine de définition un ensemble Γ ordonné par inclusion, soumis à la seule restriction que l'ordre est inductif, c'est-à-dire que Γ admet le vide comme élément, ainsi que les bornes supérieures de ses chaînes. Soit $\{G_i : i \in I\}$ une chaîne de Γ. Il faut vérifier les égalités

$$[A \cup \bigcup_i G_i] = \bigcup_i [A \cup G_i] ,$$
$$\langle \bigcup_i G_i \rangle = \bigcup_i \langle G_i \rangle ,$$

ce qui est évident. Ceci justifie la construction du domaine D par approximations successives.

Le théorème de Cantor suggère que l'absence de solution pour l'équation $E = \mathbf{Z} \cup (E \to E)$ vient de ce que l'ensemble $(E \to E)$ est "trop grand" par rapport à l'ensemble E. Il convient donc d'en considérer un sous-ensemble, et plus particulièrement le sous-ensemble $[E \to E]$. On sait déjà que cette restriction peut éliminer l'objection des cardinaux incompatibles; en particulier, l'ensemble des applications continues de $\mathbf{R}$ dans $\mathbf{R}$ est équipotent à $\mathbf{R}$ (théorème 2.17). On sait aussi que, si E est un domaine "acceptable", $[E \to E]$ le sera aussi (théorème 5.10). Néanmoins, il n'est pas évident de

[24] $\beta = \alpha + 1$ si on accepte l'hypothèse du continu généralisée.

prouver que l'opérateur ϕ défini par l'égalité $\phi(E) = A \cup [E \to E]$ est continu; le point délicat est de déterminer l'ensemble sur lequel ϕ est défini, ainsi qu'un ordre inductif adéquat sur cet ensemble. Sans entrer dans les détails techniques assez compliqués, on peut donner une idée intuitive du domaine de référence $\mathcal{D}$ sur lequel ϕ est défini. Un élément de ce domaine peut être un élément de A ou un ensemble de couples dont les origines sont deux à deux distinctes. Les éléments de ces couples sont soumis aux mêmes conditions que les éléments de D. Si A est infini, la représentation ensembliste d'un élément de $\mathcal{D}$ pourra être infinie, mais le niveau de parenthèse de tout élément de la représentation sera fini. Si on adopte une représentation arborescente, les arbres pourront être infinitaires (et donc infinis), mais la longueur de chaque branche restera finie. De plus, si A est un domaine étendu, par exemple $\mathbf{N}^+$, la relation d'ordre sur $\mathcal{D}$ s'interprétera comme "moins défini que". On aura $d \sqsubseteq e$ si l'arbre représentant d s'obtient par élagage de l'arbre représentant e. On peut montrer en formalisant cette démarche que les "équations de domaine"

$$\begin{aligned} D &= [D \to D], \\ D &= A \cup [D \to D] \end{aligned}$$

admettent des solutions, et construire ces solutions, mais cela sort du cadre de ce chapitre introductif. Le développement très informel que nous avons esquissé suffit pourtant à mettre en évidence un problème inhérent aux domaines fonctionnels récursifs employés en sémantique dénotationnelle. Puisque l'opérateur associé à l'équation $D = A \cup [D \to D]$ est continu, on peut calculer son plus petit point fixe par la technique habituelle. On pose donc

$$\begin{aligned} D_0 &= A, \\ D_{n+1} &= A \cup [D_n \to D_n], \\ D &= \bigcup_n D_n . \end{aligned}$$

Considérons alors la fonction f qui à tout élément de D associe l'élément $a \in A$. Cette fonction constante devrait a priori faire partie du domaine sémantique, mais elle n'appartient pas à D. En effet, toute fonction appartenant à D est de type $D_n \to D_n$ pour un certain n et n'admet donc pas D pour domaine.

La raison de cette lacune est que D est le plus petit point fixe de l'opérateur ϕ, alors que le domaine visé est un point fixe plus grand. Jusqu'ici, tout ensemble ou fonction défini par point fixe devait être constructible; pour cette raison, le point fixe cherché était systématiquement le moins défini, donc le plus petit. Dans le cas présent au contraire, le domaine sémantique ne doit pas être construit; c'est simplement un ensemble mathématique destiné à donner une signification à des objets qui, eux, sont construits. La différence est assez claire : on peut construire un programme de calcul de la factorielle, mais on ne construit pas directement la fonction factorielle elle-même, qui est un objet infini. On recourra donc parfois, en sémantique dénotationnelle, à des techniques de construction de domaine plus élaborées que la technique habituelle.

5.6. Application à la programmation logique

5.6.1. *Introduction*

Ce livre a illustré une série d'applications cruciales de la logique à l'informatique. La logique semble faire partie des bases mêmes de l'informatique; elle donne une sémantique aux langages de programmation et une interprétation aux programmes. On peut aller plus loin et considérer qu'un programme est un système axiomatique particulier: exécuter un programme revient à écrire (automatiquement) une preuve dans ce système. Nous allons à présent développer les bases de cette technique de programmation, actuellement très utilisée en intelligence artificielle.

Revenons encore à la définition récursive de la factorielle

(5.5) $\quad n! \;=\; \text{if } n = 0 \text{ then } 1 \text{ else } n * (n-1)!\,.$

Ecrire le développement

$$3! = 3 * 2! = 3 * 2 * 1! = 3 * 2 * 1 * 0! = 6$$

est à la fois calculer et raisonner. On observe sur cet exemple que le calcul n'est rien d'autre qu'un raisonnement particulier, à partir d'un ensemble de faits et de règles de base. La "base de connaissance" nécessaire au calcul de la factorielle d'un entier naturel donné se compose, d'une part, de la règle (5.5), éventuellement précédée de la quantification $(\forall n \in \mathbf{N})$, et, d'autre part, d'un ensemble de faits et de règles concernant l'arithmétique élémentaire et notamment la multiplication. On voudrait ici expliciter la première composante, concernant spécifiquement la factorielle, et faire abstraction de la seconde. On a une première décomposition en

$$0! = 1\,,$$
$$n \neq 0 \;\Rightarrow\; n! = n * (n-1)!\,.$$

La deuxième ligne fait intervenir l'arithmétique élémentaire. Pour favoriser le découplage, il est commode d'utiliser des prédicats plutôt que des fonctions; une première transformation conduit à

$$0! = 1\,,$$
$$(n \neq 0 \;\wedge\; n{-}1{=}x \;\wedge\; x!{=}y \;\wedge\; n * y{=}z) \;\Rightarrow\; n! = z\,.$$

Avec des notations évidentes et entièrement prédicatives, on écrira plutôt

$$f(0,1)\,,$$
$$(ne(n,0) \;\wedge\; s1(n,x) \;\wedge\; f(x,y) \;\wedge\; m(n,y,z)) \;\Rightarrow\; f(n,z)\,.$$

La seconde ligne est vraie (quantification implicite) pour tous les nombres n, x, y et z. On interprète naturellement $ne(a,b)$ en $a \neq b$, $f(a,b)$ en $a! = b$, $s1(a,b)$ en $a - 1 = b$ et $m(a,b,c)$ en $a * b = c$. Ce *programme logique* de calcul de la factorielle *définit* le prédicat f en fonction des prédicats ne, $s1$ et m, supposés prédéfinis.

5.6.2. *Nature des programmes logiques*

On pourrait définir un programme logique comme une collection finie de formules de la théorie des ensembles, contenant un prédicat inconnu f, dont le programme est la définition (dans un sens à préciser). On peut en outre se limiter aux formules de la forme

$$(\forall x_1 \in X_1) \cdots (\forall x_n \in X_n)\,[(H_1 \wedge \cdots \wedge H_p) \Rightarrow C]\,,$$

où les H_i sont les *hypothèses* et C est la *conclusion*. Le cas particulier $p = 0$ n'est pas exclu, de même qu'en mathématique on utilise des axiomes, qui sont des théorèmes de base, sans hypothèses préliminaires.

La sémantique d'un tel programme est l'ensemble des conséquences logiques de ce programme ou, plus spécifiquement, les conséquences "ayant un rapport direct" avec le prédicat f. En pratique cependant, les domaines $X_1, \ldots, X_n$ ne sont pas connus avec précision. Ils peuvent aussi être fort complexes, ce qui rend aléatoire la détermination de la sémantique du programme.

L'approche classique consiste à omettre l'information relative aux domaines et à assimiler un programme logique à une collection de formules du calcul des prédicats. La sémantique des programmes logiques fait alors usage d'un "domaine générique", appelé "domaine de Herbrand", qui a été défini et étudié dans le paragraphe I.6.6. Le théorème de Herbrand permet, dans une certaine mesure, de transporter sur d'autres domaines des conclusions obtenues à propos du domaine de Herbrand.

Un moyen plus efficace, au moins en théorie, pour s'affranchir des difficultés conceptuelles et techniques liées aux domaines est d'assimiler un programme logique à un ensemble (souvent infini) de formules du calcul des propositions. Le programme logique relatif à la factorielle comporte alors la proposition $f(0,1)$, ainsi qu'une infinité de formules telles que

$$(ne(3,0) \wedge s1(3,2) \wedge f(2,2) \wedge m(3,2,6)) \Rightarrow f(3,6)\,,$$
$$(ne(23,0) \wedge s1(23,69) \wedge f(69,12) \wedge m(23,12,2)) \Rightarrow f(23,2)\,.$$

Toutes ces formules sont interprétées comme vraies. La première traduit un raisonnement correct et utile: "Si 3 est plus grand que 0, si $3 - 1 = 2$, si la factorielle de 2 est 2 et si $3 * 2 = 6$, alors la factorielle de 3 est 6". La seconde formule traduit un raisonnement correct mais inutile, puisque certaines hypothèses sont (interprétées comme) fausses.

En ce qui concerne les prédicats auxiliaires, on supposera soit qu'ils sont interprétés ($ne(23,0)$ est synonyme de $\mathbf{V}$ et $m(23,12,2)$ est synonyme de $\mathbf{F}$), soit qu'ils font eux-mêmes l'objet de programmes logiques qui les définissent.

Remarques. Nous assimilons une formule telle que $f(3,6)$ à une proposition parce que, dans le présent contexte, elle n'a d'autre propriété que d'être vraie ou fausse. Le but de la rédaction d'un programme logique est sans doute la définition d'un prédicat f mais, comme nous l'avons vu au paragraphe I.6.4, il est parfaitement licite d'assimiler des prédicats complètement instanciés à des propositions.

La sémantique du programme sera l'interprétation qui rend vraies les conséquences logiques du programme, et fausses les autres formules. Une interprétation sera représentée par l'ensemble des propositions élémentaires qu'elle rend vraies. La sémantique du programme logique relatif à la factorielle comportera donc les propositions

$$f(0,1), f(1,1), f(2,2), f(3,6), \ldots$$

et éventuellement les propositions

$$m(3,4,12), s1(67,66), \ldots$$

mais ne comportera pas les propositions

$$f(0,2), f(3,5), m(4,4,4), \ldots$$

Exécuter le programme logique "factorielle" sur la *question* $B = f(12,6775)$ consiste à tester si la proposition B est conséquence du programme. Si c'est le cas, on en déduira[25] que la factorielle de 12 est 6 775 (ce n'est pas le cas, évidemment).

5.6.3. *Programmes logiques abstraits*

Etant donné un domaine de référence Π, c'est-à-dire un ensemble de propositions, un programme logique L est un ensemble de formules du type

$$(5.6) \qquad (p_1 \wedge p_2 \wedge \cdots \wedge p_n) \Rightarrow p_0 ,$$

où $p_0, p_1, \ldots, p_n$ sont des éléments de Π.
La formule 5.6 peut se récrire

$$(5.7) \qquad p_0 \vee \neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n .$$

Il s'agit donc d'une *clause*, c'est-à-dire d'une disjonction de littéraux, admettant exactement un littéral positif (p_0) et un nombre n quelconque, éventuellement nul, de littéraux négatifs.[26]

La sémantique du programme logique L est l'interprétation I_L telle que, pour tout $p \in \Pi$, $I_L(p)$ est vrai si et seulement si p est conséquence logique de L.

5.6.4. *Sémantique opérationnelle*

Etant donné un programme logique L sur le domaine Π, on ne cherchera généralement pas à construire globalement l'interprétation I_L (ce sera le plus

[25] à condition d'admettre l'exactitude du programme, c'est-à-dire le fait que le prédicat défini $f(x,y)$ a la même extension que l'égalité $x! = y$.

[26] On appelle *clause de Horn* toute clause ayant au plus un littéral positif. Une clause de Horn est *définie* si elle contient un littéral positif, et *négative* sinon. Une clause est *unitaire* si elle contient un seul littéral. La clause vide est une disjonction d'un ensemble vide de formules, elle est donc assimilée à **F**.

souvent un objet infini), mais on peut déterminer, pour tout élément $p \in \Pi$, si $I_L(p)$ est vrai ou faux. La technique consistant à énumérer les conséquences logiques de L peut théoriquement être acceptable, mais une énumération aveugle n'a guère de chances d'être efficace. On sait cependant que pour établir

$$L \models p\,,$$

c'est-à-dire "p est conséquence logique de L", il est suffisant d'établir

$$L \cup \{\neg p\} \models \mathbf{F}\,,$$

c'est-à-dire "l'ensemble $L \cup \{\neg p\}$ est inconsistant". (La réciproque est vraie.) On a donc $I_L(p) = \mathbf{V}$ si et seulement si l'ensemble $L \cup \{\neg p\}$ est inconsistant.

Diverses techniques pour tester l'inconsistance d'un ensemble de formules propositionnelles ou prédicatives ont été présentées dans le premier volume. Les techniques les plus performantes sont basées sur le principe de résolution.[27] Dans le cadre propositionnel, ce principe se ramène à la règle d'inférence

$$\frac{p \vee A\,,\ \neg p \vee B}{A \vee B}$$

où A et B sont des clauses, ou des formules quelconques. La validité de cette règle est évidente. Si les formules $p \vee A$ et $\neg p \vee B$ sont toutes deux vraies, alors l'une au moins des deux formules A et B est vraie. (A est vrai si p est faux, B est vrai si p est vrai.) En conséquence, la formule $A \vee B$, appelée *résolvante* de $(p \vee A)$ et $(\neg p \vee B)$ par rapport à p, est vraie.[28]

L'algorithme de *résolution unitaire positive*, présenté à la figure 5.3 en style impératif informel, permet de tester assez efficacement l'inconsistance d'un ensemble de clauses de Horn propositionnelles.

$\{S := S_0\}$
Tant que $\mathbf{F} \notin S$ faire
    choisir p et c tels que
        – p est une clause unitaire positive de S,
        – c est une clause de S contenant $\neg p$;
    $r := c \setminus \{\neg p\}$;
    $S := (S \setminus \{c\}) \cup \{r\}$.

Figure 5.3 *Résolution unitaire positive*

Dans cet algorithme, une clause est assimilée à l'ensemble des littéraux dont elle est la disjonction. Ceci permet de parler de l'appartenance d'un littéral à une clause.[29] La clause vide (ensemble vide de littéraux) est naturellement

[27] La résolution a été étudiée dans le premier volume, mais le présent exposé est autonome.

[28] La disjonction est commutative; les littéraux p et $\neg p$ ne doivent donc pas nécessairement apparaître en tête des clauses.

[29] Cette assimilation est classique; elle présente deux dangers qu'un peu d'attention permet d'éviter. Tout d'abord, un littéral ℓ est aussi une clause, ce qui revient à "confondre"

notée **F**. L'algorithme comporte un composant non déterministe de choix. Si plusieurs couples (p, c) peuvent être sélectionnés à un moment donné, un couple arbitraire acceptable est effectivement choisi. Remarquons aussi qu'une exécution de cet algorithme peut s'arrêter *normalement*, quand la condition $\mathbf{F} \in S$ devient vraie, ou s'arrêter *anormalement*, si aucun choix n'est possible pour p et c, ou encore ne jamais s'arrêter.

Remarque. La clause $r = c \setminus \{\neg p\}$ est la résolvante des clauses c et p par rapport à p; c'est un cas particulier d'application du principe de résolution rappelé plus haut, où la formule A est la clause vide et où la formule B est une clause de Horn.

L'algorithme donne lieu aux théorèmes suivants.

Théorème 5.30. Si S_0 est un ensemble fini consistant de clauses de Horn propositionnelles, alors toutes les exécutions de l'algorithme se terminent anormalement. Si S_0 est un ensemble fini inconsistant de clauses de Horn propositionnelles, alors toutes les exécutions de l'algorithme se terminent normalement.

Démonstration. Une exécution de l'algorithme est entièrement spécifiée par la suite $(S_0, S_1, \ldots)$ des valeurs de S à l'entrée de la boucle. Deux éléments consécutifs S_n et S_{n+1} ne diffèrent que par une seule clause. On a par exemple

$$\begin{aligned} S_n &= \{\ldots, p, \ldots, \neg p \vee \mathit{res}, \ldots\}, \\ S_{n+1} &= \{\ldots, p, \ldots, \quad \mathit{res} \quad, \ldots\}. \end{aligned}$$

Le symbole *res* désigne ici une clause de Horn. La formule

$$[\alpha \wedge p \wedge \beta \wedge (\neg p \vee \gamma) \wedge \delta] \equiv [\alpha \wedge p \wedge \beta \wedge \gamma \wedge \delta] \tag{5.8}$$

est une tautologie quelles que soient les formules α, β, γ et δ. On en déduit que S_n et S_{n+1} admettent exactement les mêmes modèles. On voit aussi que S_{n+1} comporte un littéral de moins que S_n (le décompte se fait répétitions comprises). Cela montre que si S_0 est fini et compte n littéraux, la longueur maximale des exécutions est $n + 1$; la valeur finale S_f de S admet exactement les mêmes modèles que la valeur initiale S_0. L'ensemble final S_f peut être de deux types. Soit il contient la clause vide **F** et est visiblement inconsistant, soit il se compose exclusivement de clauses unitaires positives et de clauses non unitaires et non vides, ne comportant la négation d'aucune proposition apparaissant comme clause unitaire positive (dans S_f). Dans ce dernier cas, S_f admet le modèle M suivant : $M(p) = \mathbf{V}$ si p est une clause unitaire positive de S_f et $M(p) = \mathbf{F}$ sinon. Cela achève la démonstration.

Remarque. Le nombre total de littéraux (avec répétitions) compris dans un ensemble de clauses de Horn est une bonne mesure de la taille de l'ensemble.

---

l'élément ℓ et le singleton $\{\ell\}$. D'autre part, un ensemble de formules est vrai pour une interprétation quand tous les éléments de l'ensemble sont vrais pour cette interprétation. La notion ensembliste de clause introduit une exception à cette règle : une clause est vraie pour une interprétation quand au moins un de ses éléments est vrai pour cette interprétation.

La résolution unitaire positive est d'efficacité linéaire en cette mesure, ce qui est très satisfaisant. Signalons néanmoins que la mise en œuvre de cette technique se fera par le biais d'une version déterministe de l'algorithme, ce qui dégrade son efficacité. Pour analyser un algorithme non déterministe, on suppose généralement que les choix sont faits instantanément et correctement; une réalisation déterministe sera alourdie par une procédure de recherche destinée à faire ces choix. Pour la résolution, on utilisera le plus souvent une double recherche linéaire dans la liste des clauses unitaires positives et celle des autres clauses. L'essentiel est que l'efficacité de l'algorithme reste polynomiale, ce qui est le cas ici.

Théorème 5.31. Si S_0 est un ensemble infini inconsistant de clauses de Horn propositionnelles, alors au moins une exécution de l'algorithme se termine normalement et aucune ne se termine anormalement. Si S_0 est un ensemble infini consistant de clauses de Horn propositionnelles, alors aucune exécution de l'algorithme ne se termine normalement.

Démonstration. Tout ensemble infini inconsistant admet un sous-ensemble fini inconsistant; en limitant les choix de clauses à ce sous-ensemble, on obtient (au moins) une exécution finie s'arrêtant normalement. D'autre part, pour toute exécution finie, S_f et S_0 sont simultanément consistants ou inconsistants. L'argument déterminant la consistance ou l'inconsistance de S_f reste valable quand S_f est infini.

Remarque. Pour certains ensembles S_0, nécessairement infinis et consistants, il n'existe que des exécutions infinies. C'est le cas de l'ensemble

$$\{p_0, \neg p_0 \vee p_1, \neg p_1 \vee p_2, \ldots\}$$

défini sur le domaine $P = \{p_i : i \in \mathbf{N}\}$ et admettant un modèle unique M, tel que $M(p_i) = \mathbf{V}$ pour tout $i \in \mathbf{N}$.

En appliquant l'algorithme de résolution unitaire positive à l'ensemble $S_0 = L \cup \{\neg p\}$ on détermine si $I_L(p) = \mathbf{V}$ ou non; on a obtenu une *sémantique opérationnelle* pour la programmation logique.

5.6.5. *Sémantique de point fixe*

La sémantique I_L d'un programme logique L défini sur le domaine Π est, formellement, un élément de l'ensemble fonctionnel $\mathcal{I}_L = (P \to \{\mathbf{F}, \mathbf{V}\})$. L'ensemble $\{\mathbf{F}, \mathbf{V}\}$, ordonné par implication, est inductif puisqu'il est fini. L'ordre canonique induit sur l'ensemble $\mathcal{I}_L$ est le suivant :

$$(I \preceq J) =_{def} (\forall p \in \Pi)\,(I(p) \Rightarrow J(p))\,.$$

Autrement dit, une interprétation I est "plus petite" qu'une interprétation J si toute proposition vraie pour I l'est également pour J. On vérifie facilement que toute interprétation est continue, et donc que l'ensemble $\mathcal{I}_L$ est le domaine $[P \to \{\mathbf{F}, \mathbf{V}\}]$. On écrit souvent $\mathcal{I}$ au lieu de $\mathcal{I}_L$.

On peut aussi, et c'est encore plus commode, assimiler $\mathcal{I}$ à l'ensemble $\mathcal{P}(\Pi)$. Dans ce cas, $p \in I$ signifie $I(p) = \mathbf{V}$ et la relation d'ordre est simplement l'inclusion. Rappelons que, pour tout ensemble Π, l'ensemble $\mathcal{P}(\Pi)$ muni de l'inclusion est un treillis complet et donc un ordre inductif.

Puisque I_L est l'ensemble des propositions conséquences logiques de L, on devine que I_L sera la limite d'une suite d'ensembles $(I_0, I_1, I_2, \ldots)$ telle que I_0 soit l'ensemble des conséquences immédiates de L, et I_{n+1} l'ensemble des propositions qui se déduisent de I_n, via L, en une étape. La théorie du point fixe permet, comme d'habitude, de formaliser cette démarche.

Les ensembles Π et L étant fixés, on définit sur le domaine $\mathcal{I}$ une transformation $\mathcal{T}_L$ (ou simplement $\mathcal{T}$), par l'égalité

$$\mathcal{T}(I) \;=\; \bigcup_{c \in L} \{p \in \Pi : c^+ = \{p\} \wedge c^- \subset I\}\,.$$

Dans cette notation, I est une interprétation, c'est-à-dire un sous-ensemble de Π; c est une clause de Horn définie, assimilée à un ensemble comportant exactement un élément de Π, formant le singleton c^+, et un nombre fini quelconque de négations d'éléments de Π, ces éléments formant l'ensemble c^-. Le programme logique L est un ensemble éventuellement infini de telles clauses.

Une proposition p appartient à $\mathcal{T}(I)$ s'il existe une clause

$$c \;=\; p \vee \neg q_1 \vee \ldots \vee \neg q_n$$

de L telle que $q_1, \ldots, q_n \in I$. (Le cas $n = 0$ n'est pas exclu.) Avec un léger abus de notation, on peut écrire

$$\mathcal{T}(I) \;=\; \bigcup_{c \in L} \{p \in \Pi : c = \{p, \neg q_1, \ldots, \neg q_n\} \wedge q_1, \ldots, q_n \in I\}\,.$$

On a les théorèmes suivants.

Théorème 5.32. Pour toute interprétation I, on a $I(L) = \mathbf{V}$ si et seulement si $\mathcal{T}(I) \subset I$.
La démonstration est évidente. Rappelons que $I(L) = \mathbf{V}$ signifie que chaque clause de L est vraie pour l'interprétation I.

Théorème 5.33. La fonction $\mathcal{T}$ est croissante.
La démonstration est évidente.

Théorème 5.34. La fonction $\mathcal{T}$ est continue.
Démonstration. On a, pour toute chaîne $\mathcal{C}$ d'interprétations, les égalités suivantes :

$$\begin{aligned}
\mathcal{T}(\bigcup \mathcal{C}) &= \bigcup_{c \in L} \{p \in \Pi : [p \in c \wedge c^- \subset \bigcup \mathcal{C}]\}, \\
&= \bigcup_{c \in L} \{p \in \Pi : [p \in c \wedge \bigvee_{I \in \mathcal{C}} (c^- \subset I)]\}, \\
&= \bigcup_{c \in L} \{p \in \Pi : \bigvee_{I \in \mathcal{C}} [p \in c \wedge c^- \subset I]\}, \\
&= \bigcup_{c \in L} \bigcup_{I \in \mathcal{C}} \{p \in \Pi : [p \in c \wedge c^- \subset I]\}, \\
&= \bigcup_{I \in \mathcal{C}} \bigcup_{c \in L} \{p \in \Pi : [p \in c \wedge c^- \subset I]\}, \\
&= \bigcup_{I \in \mathcal{C}} \mathcal{T}(I), \\
&= \bigcup \{\mathcal{T}(I) : I \in \mathcal{C}\}.
\end{aligned}$$

Le seul point délicat est la deuxième égalité. L'implication

$$\ell \in \bigcup \mathcal{C} \Rightarrow \bigvee_{I \in \mathcal{C}} (\ell \in I)$$

n'est généralement pas vraie si $\mathcal{C}$ n'est pas une chaîne (ou un ensemble dirigé) pour l'inclusion. D'autre part, l'implication

$$D \subset \bigcup \mathcal{C} \Rightarrow \bigvee_{I \in \mathcal{C}} (D \subset I)$$

n'est généralement pas vraie si l'ensemble D est infini. (Les deux implications réciproques sont toujours vraies.)

Le théorème suivant donne diverses formes de la sémantique associée à un programme logique.

Théorème 5.35. La sémantique I_L d'un programme logique L est donnée par les expressions égales suivantes :

1. $\{p \in \Pi : L \models p\}$
2. $\{p \in \Pi : L \cup \{\neg p\} \models \mathbf{F}\}$
3. $\{p \in \Pi : L \vdash p\}$
4. $\{p \in \Pi : L \cup \{\neg p\} \vdash \mathbf{F}\}$
5. $\bigcap \{I \in \mathcal{I}_L : \mathcal{T}_L(I) = I\}$
6. $\bigcap \{I \in \mathcal{I}_L : \mathcal{T}_L(I) \subset I\}$
7. $\bigcap \{I \in \mathcal{I}_L : I(L) = \mathbf{V}\}$
8. $\mathcal{T}_L \uparrow \omega$

La démonstration est simple et laissée au lecteur. L'expression $\mathcal{E} \vdash A$ signifie que la formule A est dérivable de l'ensemble $\mathcal{E}$ de formules, au moyen d'un

système de déduction approprié. Ce système est ici, au choix, la résolution générale ou la résolution unitaire positive.[30]
Il était évident a priori que tout modèle de L devait vérifier les propositions conséquences logiques de L. La théorie du point fixe donne lieu à un résultat plus fort : l'ensemble des propositions conséquences logiques de L est le plus petit modèle de L, c'est-à-dire l'intersection de tous les modèles de L. Cette propriété ne serait pas vraie pour un ensemble S de clauses quelconques. Par exemple, le singleton $\{p \vee q\}$ admet les trois modèles $\{p\}, \{q\}$ et $\{p, q\}$, mais l'intersection de ces modèles est $\emptyset$, qui n'est pas un modèle de $\{p \vee q\}$.

5.6.6. *Mise en œuvre de la programmation logique*

La programmation logique est conceptuellement très simple, mais son usage est resté limité en pratique parce qu'elle est difficile à mettre en œuvre. On distingue deux difficultés essentielles. D'une part, il est nécessaire de représenter de manière finie des ensembles de faits et de règles qui sont quasi toujours infinis et, d'autre part, on doit utiliser une version déterministe de l'algorithme de résolution, pourtant fondamentalement non déterministe.

Les solutions apportées en pratique à ces deux problèmes sont techniquement compliquées mais leurs aspects logiques, qui seuls nous intéressent ici, sont assez simples.

5.6.6.1. *Programmation logique prédicative*

L'exemple de la factorielle nous montre comment une formule de la logique prédicative, en l'occurrence, la formule

$$\forall n \forall x \forall y \forall z \, [(ne(n,0) \wedge s1(n,x) \wedge f(x,y) \wedge m(n,y,z)) \Rightarrow f(n,z)] \,,$$

remplaçait une infinité de formules propositionnelles. On pourra, en toute généralité, inclure une formule prédicative dans un programme logique, à condition que cette formule soit équivalente à un ensemble de clauses de Horn définies. Une formule telle que $\forall x \, P(x)$ sera acceptable, mais une formule telle que $\forall x \, [P(x) \vee Q(x)]$ ne le sera pas : la formule $P(a) \vee Q(a)$ n'est pas une clause de Horn, puisqu'elle est la disjonction de deux littéraux positifs. On n'acceptera pas non plus une formule du type $\exists x \, P(x)$, équivalente à une disjonction infinie. On n'acceptera pas davantage $p \vee \exists x \neg P(x)$, équivalente à une "clause de Horn infinie". (Le caractère fini des clauses est requis, en particulier, pour prouver la continuité de la fonctionnelle associée à un programme logique.) Sans entrer dans les détails, signalons qu'il suffit d'autoriser les formules obtenues en quantifiant universellement des clauses de Horn définies dont les atomes

[30] La résolution générale est un système de déduction complet; la résolution unitaire positive n'est complète que pour les clauses de Horn.

sont des applications de prédicats à des termes constants ou variables. Nous appellerons "clauses de Horn définies prédicatives" les formules de ce type.

Considérons deux telles formules A et B. Supposons aussi que ces formules soient équivalentes aux ensembles de formules propositionnelles $\mathcal{A}$ et $\mathcal{B}$. Soit alors $R(A,B)$ l'ensemble des couples (a,b), $a \in \mathcal{A}$, $b \in \mathcal{B}$, tels que a et b admettent une résolvante r. Cet ensemble peut fort bien être infini, de même que l'ensemble $\mathcal{C}$ des résolvantes. Le point crucial de la programmation logique réelle est l'existence d'une formule acceptable C équivalente à l'ensemble $\mathcal{C}$.[31] L'algorithme d'unification de Robinson (§ I.6.4) permet d'obtenir C à partir de A et B. Il existe des versions raisonnablement efficaces de cet algorithme.

En programmation logique abstraite, la question p soumise au programme L est une simple proposition. Si L est le programme "factorielle" introduit plus haut, la question p pourrait être $f(3,6)$ (question "succès") ou $f(3,7)$ (question "échec"). Cela n'est pas très utile en pratique. La question que le programmeur-utilisateur se pose ne sera pas "la factorielle de 3 est-elle 6 ?" mais plutôt "quelle est la factorielle de 3 ?" ou encore, "pour quelle valeur de x a-t-on $f(3,x)$?". La question sera donc $\exists x\, f(3,x)$; sa négation, appelée *but*, est une clause de Horn négative quantifiée universellement, à savoir $\forall x\, (\neg f(3,x))$.

En conclusion, l'usage de la logique prédicative permet d'éliminer l'obstacle lié aux ensembles infinis de clauses. Les programmes logiques réels seront des ensembles finis de clauses de Horn définies prédicatives. La question sera aussi une formule prédicative, dont la négation (le but correspondant) est une clause de Horn négative.

5.6.6.2. *Programmation logique déterministe abstraite*

Pour exécuter le programme logique (abstrait) L sur la question p, on applique l'algorithme de résolution unitaire positive au départ de l'ensemble $L \cup \{\neg p\}$. Cet algorithme implique, à chaque étape, un double choix de clauses. Ce non-déterminisme ne serait pas gênant si l'ensemble L était fini, mais ce n'est jamais le cas en pratique, d'où il est capital d'utiliser une stratégie de choix appropriée. Un premier pas dans cette direction consiste à remplacer la résolution unitaire positive (figure 5.3) par une variante appelée *résolution d'entrée*,[32] illustrée à la figure 5.4. Cette variante permet de traiter simultanément plusieurs questions, ou plus exactement la conjonction de plusieurs questions, notées ici $g_1, \ldots, g_n$. La négation de cette conjonction est le but G_0, une clause de Horn négative. L'algorithme utilise une variable G, dont la valeur est toujours une clause de Horn négative; initialement, on a $G = G_0 = \{\neg g_1, \ldots, \neg g_n\}$.

---

[31] Il est parfois nécessaire de considérer un ensemble fini $\{C_1, \ldots, C_n\}$ de formules plutôt qu'une formule unique C.

[32] *Input resolution* en anglais.

$\{G = G_0\}$
Tant que $G \neq \mathbf{F}$ faire
choisir p et c tels que
- $\neg p \in G$,
- $c \in L$ et $p \in c$;
$G := (G \setminus \{\neg p\}) \cup (c \setminus \{p\})$.

Figure 5.4 *Résolution d'entrée*

La résolution d'entrée correspond à une stratégie de preuve "à l'envers", très fréquemment employée en mathématique. On part de la conclusion à atteindre, et on cherche un théorème dont la thèse est cette conclusion. On cherche alors à prouver les hypothèses du théorème employé (ce sont des "sous-buts") en appliquant la même stratégie. On continue ainsi, en gérant une liste de sous-buts. Un sous-but peut être éliminé s'il est la négation d'un axiome (thèse sans hypothèse), ce qui permet éventuellement la terminaison du processus de preuve, dès que la liste des sous-buts est vide. Dans l'algorithme, la variable G est l'ensemble des sous-buts; rappelons que $G = \emptyset$ est ici synonyme de $G = \mathbf{F}$. On peut prouver que, dans le cas fini, la résolution d'entrée est équivalente à la résolution unitaire positive.

5.6.6.3. *Programmation logique déterministe réelle*

En pratique, les programmes logiques sont formés de clauses de Horn prédicatives; l'exécution d'un programme sur un but donné combine la résolution d'entrée et l'unification. La résolution d'entrée est rendue déterministe en considérant que les clauses et aussi les littéraux négatifs composant le but forment des listes ordonnées et non des ensembles. Dans l'algorithme, la proposition p choisie sera toujours celle dont l'opposé est le premier élément de la liste-but; la clause c choisie sera la première clause applicable de la liste-programme. Une clause est applicable si, au prix d'une unification éventuelle, son littéral positif est p.

Les techniques d'exécution de programmes logiques réels sont du domaine de l'informatique; nous évoquerons ici seulement un exemple simple. La concaténation de listes peut se définir par un prédicat *append* à trois arguments, tel que $append(X, Y, Z)$ est vrai si Z est la concaténation des listes X et Y. On suppose prédéfini l'opérateur "cons", qui introduit un élément en tête d'une liste, et on note cet opérateur par le symbole | (utilisé de manière infixée).

La concaténation est définie par le programme logique suivant :

$append([\,], Ys, Ys)$,
$append([X|Xs], Ys, [X|Zs]) \vee \neg\ append(Xs, Ys, Zs)$.

On peut se servir de ce programme pour décomposer une liste $[a, b]$ en un préfixe et un suffixe; le but correspondant sera

$\neg$ *append*(Xs, Ys, $[a, b]$).

Les variables X, Xs, Ys et Zs sont, de manière implicite, quantifiées universellement. Chaque clause est indépendante des autres, et l'identité des noms entre deux variables intervenant dans des clauses distinctes est purement accidentelle. La négation du but est la question, c'est-à-dire, existe-t-il des listes Xs et Ys telles que la concaténation de ces listes soit la liste $[a, b]$? Un système de programmation logique établit des preuves constructives pour les assertions (questions) de ce type.

La "trace" de l'exécution, sur un système réel (Prolog), est :

```
?- append(X,Y,[a,b,c]).
    --> X = [] , Y = [a,b] ;
    --> X = [a] , Y = [b] ;
    --> X = [a,b] , Y = [] ;
    --> no
```

Elle exprime la découverte de trois solutions au problème posé et l'impossibilité d'en donner une quatrième.

La recherche s'organise sous forme d'un arbre construit et parcouru de la gauche vers la droite, en ne passant d'une branche à la suivante que quand l'extrémité de cette branche a été atteinte. Dans le cas présent, l'arbre sera binaire parce que le programme comporte deux clauses (figure 5.5). En tout nœud, la branche de gauche correspond au choix de la première clause, et la branche de droite au choix de la seconde. Nous donnons une description plus détaillée de cette recherche, correspondant au parcours de l'arbre. Pour rappeler l'indépendance mutuelle des clauses et du but, les variables des clauses sont systématiquement renommées à chaque unification; l'exposant dont on les affecte correspond au niveau d'utilisation dans l'arbre.

La première solution s'obtient en unifiant le but initial **0** et la première clause (dont les variables sont affectées de l'exposant 1). La substitution adéquate unifie, c'est-à-dire rend identique, au signe près, l'unique littéral (négatif) du but et l'unique littéral (positif) de la clause. Cette substitution, indiquée sur la branche 0-1 de l'arbre, est Xs, Ys, $Ys^1 \leftarrow [\,], [a, b], [a, b]$. La résolution produit alors **F** (nœud **1**), ce qui indique la découverte d'une solution. Le système Prolog la décrit en donnant l'unification qui y a conduit, restreinte aux variables apparaissant dans le but, c'est-à-dire Xs, $Ys = [\,], [a, b]$.

Le système "remonte" au nœud **0** (*backtracking*, en anglais) et unifie le but correspondant au littéral positif de la seconde clause (variables affectées de l'exposant 1 : on est toujours au premier niveau de l'arbre). La substitution adéquate est $Xs, Ys, X^1, Zs^1 \leftarrow [a|Xs^1], Ys^1, a, [b]$; elle conduit au nouveau but $\neg$ *append*(Xs^1, Ys^1, $[b]$) (nœud **2**). Une deuxième unification, maintenant avec la première clause (exposant 2 : on est au deuxième niveau), conduit à **F** (nœud **3**), la substitution étant ici $Xs^1, Ys^1, Ys^2 \leftarrow [\,], [b], [b]$. La solution correspondante s'obtient en combinant les deux substitutions; on a donc

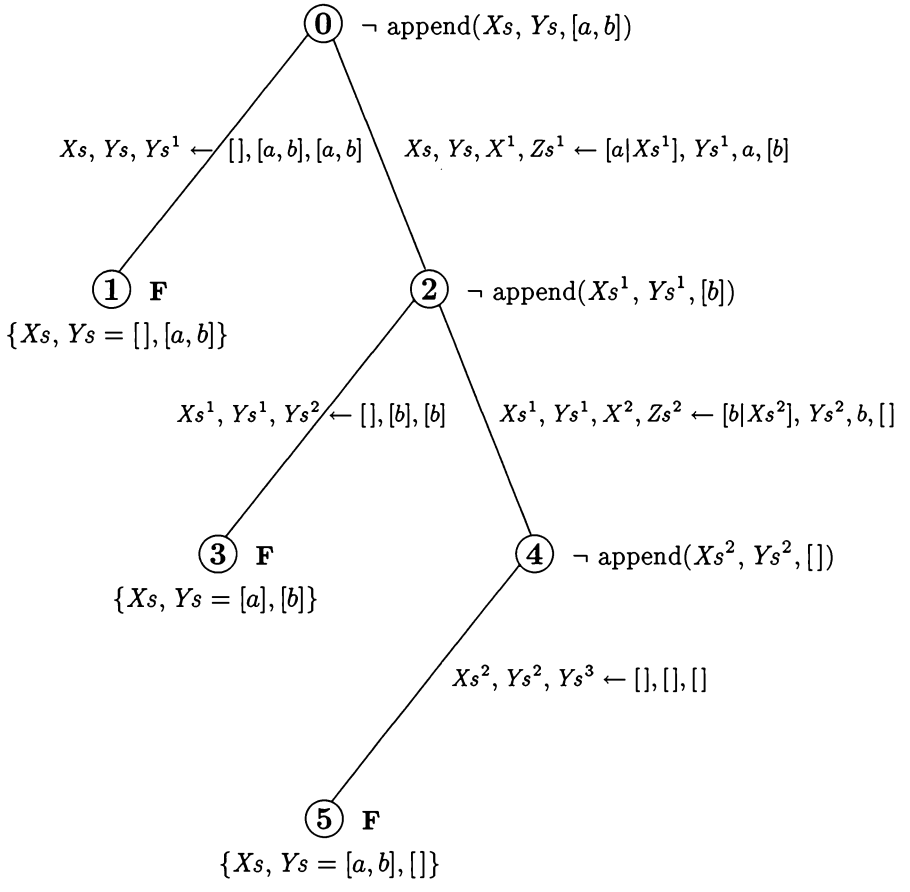

Figure 5.5 *Arbre de recherche*

$Xs, Ys = [a|Xs^1], Ys^1 = [a|[]], [b] = [a], [b]$.

Le système reconsidère alors le but précédent (remontée au nœud **2**), mais l'unifie cette fois avec la seconde clause. Le but résultant (nœud **4**) est obtenu au moyen de la substitution $Xs^1, Ys^1, X^2, Zs^2 \leftarrow [b|Xs^2], Ys^2, b, []$. Ce dernier but peut à son tour être unifié, mais seulement avec la première clause; la résolution conduit à **F** (nœud **5**), par la substitution $Xs^2, Ys^2, Ys^3 \leftarrow [], [], []$. La solution correspondante s'obtient par composition de substitutions : $Xs, Ys = [a|Xs^1], Ys^1 = [a|[b|Xs^2]], Ys^2 = [a|[b|[]]], [] = [a, b], []$.

Avertissement. Seul le premier chapitre, à vocation plus historique, a donné lieu à une étude bibliographique. Les chapitres suivants se veulent introductifs, et nous avons considéré qu'une bibliographie abondante n'était guère utile au lecteur débutant. En particulier, la lecture des textes d'origine n'est pas toujours le meilleur moyen de s'initier à un sujet donné. Elle est, en revanche, extrêmement profitable à un stade ultérieur. Nous avons repris ici la liste des références utilisées ou consultées lors de la rédaction de ce livre (la liste des références pertinentes aurait été beaucoup plus longue), ainsi que quelques ouvrages utiles au lecteur voulant approfondir ou augmenter sa culture dans les domaines traités ici.

Ce livre se veut accessible à tout lecteur pourvu d'une culture générale "normale", et d'une maîtrise suffisante de la logique classique élémentaire, acquise par exemple par la lecture du premier volume (ou d'un autre traité de logique). Néanmoins, le lecteur peut souhaiter consulter, d'une part, l'un ou l'autre ouvrage développant les mêmes sujets, sous un angle différent et, d'autre part, des livres ou articles plus spécialisés, permettant d'approfondir certains points. Nous donnons ici quelques indications, peut-être subjectives, sans aucune prétention à l'exhaustivité.

En ce qui concerne la théorie des ensembles, le classique *Naive Set Theory* de Halmos (1965), traduit en français, reste incontournable. La théorie axiomatique est présentée avec clarté et concision par Krivine (1969). De nombreux ouvrages d'introduction à la logique font une place importante à la théorie des ensembles; citons notamment le récent *Logique mathématique* de Cori et Lascar (1993). A un niveau plus spécialisé, le *Handbook of Mathematical Logic* (Barwise 1974) aborde les domaines essentiels de la logique et de la théorie des ensembles; il comporte des indications historiques et des pointeurs vers la littérature spécialisée. Enfin, le premier livre des *Eléments de Mathématique*, de l'équipe Bourbaki, est entièrement consacré à la théorie des ensembles.

Des introductions générales à l'informatique et à la programmation sont *Foundations of Computer Science* de Aho et Ullman (1992, trad. franç. 1993) et *Cours d'informatique : langages et programmation* de Scholl et al. (1993). Tout en étant déjà très complets, ces livres restent accessibles au débutant. Plus spécifiquement orientés vers la logique, mentionnons, en français, la *Théorie des programmes* de Livercy (1978) et *La construction de programmes structurés* d'Arsac (1977). En anglais, citons notamment *A discipline of programming* de Dijkstra (1976), *The science of programming* de Gries (1981) et, pour la programmation parallèle, *Parallel Program Design* de Chandy et Misra (1988).

Relativement peu d'ouvrages accessibles au débutant comportent une introduction à la théorie du point fixe et à ses applications. Outre le livre de Livercy, déjà cité, mentionnons *Mathematical Theory of Computation* de Manna (1974), *Mathematical Theory of Program Correctness* de de Bakker (1980), et *The Formal Semantics of Programming Languages* de Winskel (1993). Ce dernier ouvrage comporte une bibliographie très complète.

Bibliographie

[1] Andrews P.B., *A transfinite type theory with type variables*, North-Holland, Amsterdam, 1965.

[2] Aczel P., *Lectures on non-well-founded sets*, CSLI Lecture Notes **9**, Springer, 1987.

[3] Apt K.R., Ten years of Hoare logic, part I, *ACM Trans. Prog. Lang. Sys.* **3** (1981) 431-483.

[4] Apt K.R., Ten years of Hoare logic, part II: Nondeterminism, *Theoret. Comput. Sci.* **28** (1984) 83-109.

[5] Apt K.R., Correctness Proofs of Distributed Termination Algorithms, *ACM Trans. Prog. Lang. Sys.* **8** (1986) 388-405.

[6] Back R.J.R., A Calculus of Refinements for Program Derivations, *Acta Informatica* **25** (1988) 593-624.

[7] Back R.J.R., A Method for Refining Atomicity in Parallel Algorithms, *Lect. Notes Comput. Sci.* **366** (1989) 199-216.

[8] Back R.J.R., Kurki-Suonio R., Decentralization of Process Nets with Centralized Control, *Proc. 2nd ACM Symp. on Principles of Distributed Computing* (1983) 131-142.

[9] Back R.J.R., Kurki-Suonio R., Distributed Cooperation with Action Systems, *ACM Trans. Prog. Lang. Sys.* **10** (1988) 513-554.

[10] Barwise J. (Ed.), *Handbook of Mathematical Logic*, North-Holland, Amsterdam, 1974.

[11] Barwise J., Etchemendy J., *The Liar - An Essay on Truth and Circularity*, Oxford University Press, 1987.

[12] Best E., A Note on the Proof of a Concurrent Program, *Inform. Proc. Lett.* **9** (1979) 103-104.

[13] Boffa M., Les ensembles extraordinaires, *Bulletin de la Société mathématique de Belgique* **20** (1968) 3-15.

[14] Bolzano B., *Les paradoxes de l'infini*, introduction et traduction de H. Sinaceur, Le Seuil, Paris, 1993.

[15] Bourbaki N., *Eléments de Mathématique*, Hermann, Paris, 1954-.

[16] Boyer R.S., Moore J.S., *A computational logic*, Academic Press, London, 1979.

[17] Breuer J., *Initiation à la théorie des ensembles*, traduction et adaptation de A. Gloden, Dunod, Paris, 1961.

[18] Cantor G., *Contributions to the Founding of the Theory of Transfinite Numbers*, introduction et traduction de Ph. Jourdain, Open Court, 1915, Dover, New York, 1955.

[19] Cavaillès J., *Philosophie mathématique*, préface de R. Aron, introduction de R. Martin, Hermann, Paris, 1962.

[20] Chandy M., Misra J., An Example of Stepwise Refinement of Distributed Programs: Quiescence Detection, *ACM Trans. Prog. Lang. Sys.* **8** (1986) 326-343.

[21] Chandy M., Misra J., *Parallel Program Design: A Foundation*, Addison-Wesley, Reading (Mass.), 1988.

[22] Chierchia G., Turner R., Semantics and property theory, *Linguistics and Philosophy* **11**, (1988) 261-302.

[23] Church A., Comparison of Russell's Resolution of the Semantical Antinomies with that of Tarski, *Journal of Symbolic Logic* **41**, (1976) 747-760 (*repris dans* Martin R., 1984).

[24] Chwistek L., The Theory of Constructive Types, *Ann. Soc. Pol. Math.* **2** 9-48, **3** 92-141 (1924).

[25] Clarke E.M., Program invariants as fixed points, *Proc. 18th IEEE Symp. on Foundations of Computer Science* (1977) 18-29.

[26] Clarke E.M., Synthesis of Resource Invariants for Concurrent Programs, *ACM Trans. Prog. Lang. Sys.* **2** (1980) 338-358.

[27] Clarke E.M., Grumberg O., Browne M.C., Reasoning about networks with many identical finite-state processes, *Proc. 5th ACM Symp. on Principles of Distributed Computing* (1986) 240-248.

[28] Cocchiarella N., Review of *The philosophy of W.V. Quine*, *History and Philosophy of Logic* **10** (1989) 77-83.

[29] Cori R., Lascar D., *Logique mathématique I et II*, Masson, Paris, 1993.

[30] Cousot P., Cousot R., Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, *Proc. 4th ACM Symp. on Principles of Programming Languages* (1977) 238-252.

[31] Cousot P., Cousot R., A Language Independent Proof of the Soundness and Completeness of Generalized Hoare Logic, *Information and Computation* **80** (1989) 165-191.

[32] Cousot P., Cousot R., Automatic discovery of linear restraints among variables of a program, *Proc. 5th ACM Symp. on Principles of Programming Languages* (1978) 84-96.

[33] Crabbé M., Le calcul lambda, *Cahiers du Centre de Logique* **6**, Cabay, Louvain la Neuve (1986) 63-86.

[34] De Bakker J.W., Meertens L.G.L.T., On the completeness of the inductive assertion method, *Journ. Comput. Syst. Sci.* **11** (1975) 323-357.

[35] De Bakker J.W., *Mathematical Theory of Program Correctness*, Prentice Hall, New Jersey, 1980.

[36] Degen J.W., *Systeme der kumulativen Logik*, Philosophia Verlag, Munich, 1983.

[37] Delahaye J.P., Nouvelles solutions pour les paradoxes sémantiques, Lille, 1990.

[38] Delahaye J.P., Le réalisme en mathématiques et en physique, *Pour la Science* **159** (1991) 34-42.

[39] de Rouilhan Ph., *Frege: les Paradoxes de la Représentation*, Ed. de Minuit, Paris, 1988.

[40] Dijkstra E.W., *A discipline of programming*, Prentice Hall, New Jersey, 1976.

[41] Dijkstra E.W., A correctness proof for networks of communicating processes: a small exercise, EWD 607, Burroughs, The Netherlands, 1977.

[42] Dijkstra E.W., On-the-Fly Garbage Collection: An Exercise in Cooperation, *Comm. ACM* **21** (1978) 966-975.

[43] Dijkstra E.W., Finding the Correctness Proof of a Concurrent Program, *Lect. Notes Comput. Sci.* **69** (1979) 24-34.

[44] Dijkstra E.W., On the Interplay between Mathematics and Programming, *Lect. Notes Comput. Sci.* **69** (1979) 35-46.

[45] Dijkstra E.W. (Ed.), *Formal Development of Programs and Proofs*, Addison-Wesley, Reading (Mass.), 1990.

[46] Feferman S., Infinity in Mathematics – Is Cantor Necessary ? *Infinity in Science*, Istituto della Enciclopedia Italiana, Roma (1987) 151-209.

[47] Feijen W.H.J., Exercises in Formula Manipulation, in [45].

[48] Fraenkel A., Bar-Hillel Y., Levy A., *Foundations of Set Theory*, North-Holland, Amsterdam, 1973.

[49] Frege G., *Les fondements de l'arithmétique*, traduction et introduction de Cl. Imbert, Le Seuil, Paris, 1969.

[50] Gerth R., Transition Logic, *Proc. 16th ACM Symposium on Theory of Computing* (1984) 39-50.

[51] Gödel K., *Collected Works*, Vol. II: Publications 1938-1974, Oxford University Press, 1990, ed. by S. Feferman et al.

[52] Gribomont E.P., Stepwise refinement and concurrency: the finite-state case, *Sci. Comput. Prog.* **14** (1990) 185-228.

[53] Gribomont E.P., Concurrency without toil: a systematic method for parallel program design, *Sci. Comput. Prog.* **21** (1993) 1-56.

[54] Gries D., *The Science of Programming*, Springer, Berlin, 1981.

[55] Hallett M., *Cantorian Set Theory and Limitation of Size*, Clarendon Press, Oxford, 1984.

[56] Halmos A., *Naive Set Theory*, Springer, Berlin, 1965, 1974 (Trad. *Introduction à la théorie des ensembles*, Gauthier-Villars, Paris, 1967).

[57] Heinzmann G., *Entre intuition et analyse: Poincaré et le concept de prédicativité*, Blanchard, Paris, 1985.

[58] Heinzmann G. (Ed.), *Poincaré, Russell, Zermelo et Peano*, Blanchard, Paris, 1986.

[59] Hoare C.A.R., An axiomatic basis for computer programming, *Comm. ACM* **12** (1969) 576-583.

[60] Hoare C.A.R., Proof of a program: FIND, *Comm. ACM* **14** (1971) 39-45.

[61] Hoare C.A.R., Communicating Sequential Processes, *Comm. ACM* **21** (1978) 666-677.

[62] Hoare C.A.R., *Communicating Sequential Processes*, Prentice-Hall, New Jersey, 1985.

[63] Imbert Cl., *voir* Frege G.

[64] Kant E., *Prolégomènes à toute métaphysique future*, trad. J. Gibelin, Vrin, Paris, 1930.

[65] Keller R.M., Formal Verification of Parallel Programs, *Comm. ACM* **19** (1976) 371-384.

[66] Kripke S., Outline of a theory of Truth, *The Journal of Philosophy* **72** (1975) 690-716.

[67] Krivine J.-L., *Théorie axiomatique des ensembles*, Presses Universitaires de France, Paris, 1969.

[68] Lamport L., A New Solution of Dijkstra's Concurrent Programming Problem, *Comm. ACM* **17** (1974) 453-455.

[69] Lamport L., The 'Hoare Logic' of Concurrent Programs, *Acta Informatica* **14** (1980) 21-37.

[70] Lamport L., An Assertional Correctness Proof of a Distributed Algorithm, *Sci. Comput. Prog.* **2** (1983) 175-206.

[71] Lamport L., Schneider F.B., The 'Hoare Logic' of CSP, and All That, *ACM Trans. Prog. Lang. Sys.* **6** (1984) 281-296.

[72] Lamport L., *win* and *sin*: Predicate Transformers for Concurrency, DEC SRC Report 17 (1987).

[73] Lamport L., Control Predicates are Better than Dummy Variables for Reasoning about Program Control, *ACM Trans. Prog. Lang. Sys.* **10** (1988) 267-281.

[74] Lamport L., A Theorem on Atomicity in Distributed Algorithms, DEC SRC Report 28 (1988).

[75] Largeault J., *Logique et philosophie chez Frege*, Nauwelaerts, Paris Louvain, 1971.

[76] Leibniz G.W., Lettre pour Jean Gallois, *Sämtliche Schriften und Briefe Preussische Akademie der Wissenschaften*, Berlin A II, I (1923-) 222-229.

[77] Livercy C., *Théorie des programmes*, Dunod, Paris, 1978.

[78] Lipton R.J., Reduction: a method of proving properties of parallel programs, *Comm. ACM* **18** (1975) 717-721.

[79] Lynch N.A., Tuttle M.R., Hierarchical Correctness Proofs for Distributed Algorithms, *Proc. 6th ACM Symp. on Principles of Distributed Computing* (1987) 137-151.

[80] Maddy P., Believing the axioms I, II, *Journal of Symbolic Logic* **53** (1988) 481-511, 736-764.

[81] Martin R., *Recent Essays on Truth and the Liar Paradox*, Clarendon Press, Oxford, 1984.

[82] May D., OCCAM, *ACM Sigplan Notices* **18**.4 (1983) 69-79.

[83] Moore G.H., *Zermelo's axiom of choice: its origins, development and influence*, Springer, Berlin, 1982.

[84] Morgan C., The Specification Statement, *ACM Trans. Prog. Lang. Sys.* **10** (1988) 403-419.

[85] Morgan C., Auxiliary variables in data refinement, *Inform. Proc. Lett.* **29** (1988) 293-296.

[86] Morris J., A theoretical basis for stepwise refinement and the programming calculus, *Sci. Comput. Prog.* **9** (1987) 287-306.

[87] Morris J.M., Varieties of weakest liberal preconditions, *Inform. Proc. Lett.* **25** (1987) 207-210.

[88] Owicki S., Gries D., An Axiomatic Proof Technique for Parallel programs, *Acta Informatica* **6** (1976) 319-340.

[89] Parsons T., Type Theory and ordinary Language, *in* Mithin and Davis (eds.) *Linguistics, Philosophy and Montague Grammar*, University of Texas, Austin, 1979.

[90] Peterson G.L., Myths about the mutual exclusion problem, *Informat. Proc. Letters* **12** (1981) 115-116.

[91] Pieters J., La théorie des types de Russell et Whitehead, *Cahiers du Centre de Logique* **2**, Cabay, Louvain la Neuve (1981) 33-59.

[92] Poincaré H., Les mathématiques et la logique (1906), *in* Heinzmann (ed.) *Poincaré, Russell, Zermelo et Peano*, Blanchard, Paris, 1986.

[93] Quine W.V.O., *Set Theory and its Logic*, The Belknap Press, Cambridge, Mass., 1963.

[94] Quine W.V.O., *The Ways of Paradox and Other Essays*, Random House, New-York, 1966.

[95] Ramsey F.P., *Foundations*, Routledge & Kegan Paul, Londres, 1978.

[96] Raynal M., *Algorithms for Mutual Exclusion*, North Oxford Academic, 1986.

[97] Ricart G., Agrawala A.K., An optimal algorithm for mutual exclusion, *Comm. ACM* **24** (1981) 9-17 (corr. **24** 578).

[98] Rivenc F., *Recherches sur l'universalisme logique, Russell et Carnap*, Payot et Rivages, Paris, 1993.

[99] Rogers, R., *Mathematical logic and formalized theories*, North-Holland, Elsevier, Amsterdam, 1971.

[100] Russell B., *The Principles of Mathematics*, Allen & Unwin, Londres, 1903.

[101] Russell B., On some difficulties in the theory of transfinite numbers and order types, *et* Les paradoxes de la logique, *in* G. Heinzmann (ed.), *Poincaré, Russell, Zermelo et Peano*, Blanchard, Paris, 1986 (pp. 54-78 et 121-144).

[102] Russell B., *Ecrits de logique philosophique*, Avant-propos et traduction de J.M. Roy, P.U.F., Paris, 1989.

[103] Russell B., La logique mathématique fondée sur la théorie des types, in *Logique et fondements des mathématiques, Anthologie* (1850-1914) sous la direction de F. Rivenc et Ph. de Rouilhan, Payot, Paris, 1992.

[104] Scholl P.-C., Fauvet M.-C., Lagnier F., Maraninchi F., *Cours d'informatique : langages et programmation*, Masson, Paris, 1993.

[105] Scott D., Axiomatizing set theory, in *Axiomatic set theory, AMS Proc. of Symp. of Pure Mathematics* **13** (1974) 207-214.

[106] Saint-James E., *La programmation applicative*, Hermès, Paris, 1993.

[107] Shapiro S., *Foundations without foundationalism - A case for Second-order Logic*, Clarendon Press, Oxford, 1991.

[108] Sifakis J., A unified approach for studying the properties of transition systems, *Theor. Comput. Sci.* **18** (1982) 227-259.

[109] Tarski A., *Logique, sémantique, métamathématique*, traduit sous la direction de G.G. Granger, Armand Colin, Paris, 1972 (tome 1), 1974 (tome 2).

[110] Urquhart A., Russell's zigzag path to the ramified theory of types, in *Antinomies & Paradoxes*, pp. 82-91, Ed. Winchester I. and Blackwell K., McMaster University, 1989.

[111] Van Lamsweerde A., Sintzoff M., Formal derivation of strongly correct concurrent programs, *Acta Informatica* **12** (1979) 1-31.

[112] Vernant D., *La philosophie mathématique de Russell*, Vrin, Paris, 1993.

[113] Whitehead A.N., Russell B., *Principia mathematica*, Cambridge University Press, 1913 (1ère éd.), 1927 (2ème éd.).

[114] Wittgenstein L., *Tractatus logico-philosophicus*, 1921, Introduction de B. Russell, traduction de D.F. Pears et B.F. McGuinness, Routledge & Kegan Paul, London, 1961.

[115] Wolper P., Lovinfosse V., Verifying Properties of Large Sets of Processes with Network Invariants, *Proc. Workshop on Automatic Verification* (Grenoble, 1989).

[116] Zermelo E., Recherches sur les fondements de la théorie des ensembles, traduit et introduit par F. Longy, in *Logique et fondements des mathématiques, Anthologie* (1850-1914) sous la direction de F. Rivenc et Ph. de Rouilhan, Payot, Paris, 1992.

Index

Accessible
    graphe —, 119
Adhérence, 288
Admissible
    propriété —, 289
Affectation, 141–143, 162
    conditionnelle, 153
    gardée, 153
    multiple, 143, 174
    non déterministe, 145
    simple, 174
    vide, 143
Algorithme
    de Dijkstra, 257
    de Hoare, 184
    de Peterson, 224
    de Ricart et Agrawala, 234
    de tri, 182
    systolique, 216
Alternée
    classe —, 101
Anti-chaîne, 273
Application, 58
    bijective, 58
Assertion, 146
Asynchrone
    canal —, 233
    communication —, 233
Automate de Conway, 218
Automorphisme, 110
Axiomatique
    sémantique —, 164
Axiome
    de Church, 92
    de fondation, 31, 115
    de l'infini, 26, 98
    de la puissance, 27, 95
    de la réunion, 26, 95
    de Peano, 70
    de remplacement, 28, 85
    de renommage, 85
    de substitution, 28
    du choix, 29, 61

Barbier
    paradoxe du —, 23
Bijection, 57
Bijective
    application —, 58
Bisimulation, 121

Borne
plus grande — inférieure, 272
plus petite — supérieure, 272
meilleure —, 272

Canal, 215
asynchrone, 233
synchrone, 215
Canonique
graphe —, 119
Cardinal, 122
Chaîne, 64, 273
Champ
d'un enregistrement, 158
identificateur de —, 158
Choix
axiome du —, 29, 61
fonction de —, 61
Church
thèse de —, 196
Classe, 52, 53, 89
alternée, 101
d'équivalence, 62, 63
de Russell, 89
ordinaire, 101
propre, 92
régulière, 101
transitive, 101
universelle, 91
Clause, 324
de Horn, 324
définie, 324
négative, 324
unitaire, 324
Codomaine
d'une relation, 56
Coextensive
relation —, 56
Collection, 89
Combinateur, 173
Communication
asynchrone, 233
synchrone, 215
Compact
ensemble —, 285
Complet
treillis —, 287
Composante
d'un tableau, 156
Composition, 58
Compositionnalité, 207
Compositionnelle
sémantique —, 317
Compréhension
définition en —, 54
principe de —, 54
schéma de —, 91
Condition, 153
de terminaison, 171
de vérification, 186
initiale, 169
Conditionnelle
affectation —, 153
Connexe
graphe —, 118
Constante, 142, 163, 173, 201
Correction partielle, 154, 179
Correction totale, 155
Correspondance, 58
Couple, 56
Coupure, 86
Cyclique
processus —, 219

Décoration
d'un graphe, 119
Définition
en compréhension, 54
en extension, 53
récursive, 60, 72
schéma de —, 80
Dénombrable
ensemble —, 123
Dénotationnelle
sémantique —, 317
Dijkstra

algorithme de —, 257
Dirigé
ensemble —, 273
Domaine, 295
d'une relation, 56
dérivé, 268
primitif, 268

Elément, 52
fondamental, 52, 134, 268
maximal, 272
minimal, 272
Enregistrement, 156, 158
champ d'un —, 158
Ensemble, 52, 53, 89
équipotent, 112, 122
bien fondé, 64, 273
bien ordonné, 64, 273
compact, 285
dénombrable, 123
dirigé, 273
fermé, 283
fini, 107, 123
inductif, 69, 277
infini, 123
ouvert, 282
partie d'un —, 55
sous-ensemble, 55
type d'un—, 52
vide, 53
— vs. Classe, 52
— vs. Collection, 51, 52, 89
Environnement, 318
Equipotent
ensemble —, 112, 122
Equivalence, 59
classe d'—, 62, 63
Espace topologique, 282
Etat, 142, 162
final, 162
initial, 162
successeur, 162
terminal, 162
Etiquette, 162
Evaluation
d'une expression, 141
Exécutable
transition —, 162
Exécution, 162
pas d'—, 207
Exclusion mutuelle, 219
Expression, 141
évaluation d'une —, 141
arithmétique, 141
type d'une —, 143
Extension
d'une relation, 56
définition en —, 53
naturelle, 293
stricte, 293
Extensionnalité
principe d'—, 53
schéma d'—, 91
Extensionnelle
relation —, 117
Extrémité, 56, 162
Extralogique
symbole —, 63

Fermé
ensemble —, 283
Filtre, 274, 275, 287
Final
état —, 162
Fini
ensemble —, 107, 123
Finitaire
non-déterminisme —, 197
Fonction, 56
continue, 278
croissante, 275
de choix, 61
injective, 58
monotone, 275
partielle, 58
restriction d'une —, 56

surjective, 58
totale, 58
Fonctionnelle, 81, 290
relation —, 58
Fondamental
élément —, 52, 134, 268
Fondation
axiome de, 31, 115
Forme, 293

Gardée
affectation —, 153
Garde, 153
Genre, 268
Graphe
accessible, 119
bien fondé, 119
canonique, 119
connexe, 118
décoration d'un —, 119
Générique, 57

Hoare
algorithme de —, 184
triplet de —, 145, 147, 177

Idéal, 274, 275, 286
Identificateur, 317
de champ, 158
Image, 56
Inclusion, 54
Index
d'un tableau, 156
Inductif
ensemble —, 69, 277
Induction
principe d'—, 80
structurelle, 304
Infimum, 272
Infini
axiome de l'—, 26, 98
ensemble —, 123
paradoxe de l'—, 17, 18
Infinitaire
non-déterminisme —, 198
Initial
état —, 162
segment —, 275
Injective
fonction —, 58
Instruction
atomique, 207
Intérieur, 288
Intension
d'une relation, 56
Interblocage, 231
Intersection, 55
généralisée, 55
Invariant, 170, 171, 178
Isomorphisme, 57

Lexique, 141
Limite
ordinal —, 106
Locale
variable —, 202
Logique
typée, 268

Mémoire, 145, 162
Majorant, 270
Maximal
élément —, 272
Maximum, 270
Menteur
paradoxe du —, 33
Minimal
élément —, 272
Minimum, 270
Minorant, 270
Multiple
affectation —, 143

Naturel, 107
Nombre réel, 86
Non-déterminisme, 145, 162, 172, 197
finitaire, 197

infinitaire, 198

Opérationnelle
sémantique —, 144, 175
Oracle, 196
Ordinaire
classe —, 101
Ordinal, 100, 101
limite, 106
successeur, 106
Ordre
bien fondé, 273
bon —, 64
relation d'—, 59
total, 59
Origine, 56, 162
Ouvert
ensemble —, 282

Paradoxe
de l'infini, 17, 18
de Russell, 23, 24, 89
du barbier, 23
du menteur, 33
Paramètre, 202
formel, 202
réel, 202
Partie
d'un ensemble, 55
Partielle
fonction —, 58
Pas d'exécution, 207
Peano
axiomes de —, 70
Perceptron, 218
Peterson
algorithme de —, 224
Pile, 159
Point de contrôle, 162
Point fixe, 256, 270
Postcondition, 147
plus forte —, 150
Précondition, 147
plus faible —, 149
Prédécesseur, 118
Prédicat de contrôle, 164
Préordre
relation de —, 59
Pragmatique, 141
Principe
d'extensionnalité, 53
d'induction, 80
de compréhension, 54
de récurrence, 80
Procédure, 202
Processus, 208
cyclique, 219
Produit
d'ensembles, 61
Programme, 142
Propre
classe —, 92
Propriété
admissible, 289
d'invariance, 169
d'un programme, 169
de progrès, 169
de terminaison, 171
Puissance
axiome de la —, 27, 95

Quotient
ensembliste, 62

Réciproque
d'une relation, 57
Récurrence
principe de —, 80
Récursion, 79
Récursive
définition —, 60, 72
Récursivité, 79
Réduction
relation de —, 302
Réflexivité
d'une relation, 59
Régulière
classe —, 101

Réunion, 55
  axiome de la —, 26, 95
  généralisée, 55
Règle
  de calcul, 300
  de contrôle, 164
  de généralisation typée, 271
Relation, 56
  (anti-)réflexive, 59
  (anti-)symétrique, 59
  codomaine d'une —, 56
  coextensive, 56
  d'entrée-sortie, 164
  d'ordre, 59
  de préordre, 59
  de réduction, 302
  domaine d'une —, 56
  extension d'une —, 56
  extensionnelle, 117
  fonctionnelle, 58
  intension d'une —, 56
  réciproque, 57
  transitive, 59
Relationnelle
  sémantique —, 145, 163
Remplacement
  axiome de —, 28, 85
Renommage
  axiome de —, 85
Restriction
  d'un type, 271
  d'une fonction, 56
Ricart et Agrawala
  algorithme de —, 234
Russell
  paradoxe de —, 23, 24

Sémantique, 141
  axiomatique, 164
  compositionnelle, 317
  dénotationnelle, 317
  opérationnelle, 144, 175
  relationnelle, 145, 163
Séparation
  schéma de —, 91
Séquence, 62
Séquenceur, 206
Schéma
  d'axiome de Church, 92
  d'extensionnalité, 91
  de compréhension, 91
  de définition récursive, 80
  de séparation, 91
Section
  critique, 219
  non critique, 219
Segment initial, 86, 275
Signature, 63
Singleton, 68
Sous-ensemble, 55
Sous-tableau, 183
Stationnaire
  suite —, 273
Substitution
  axiome de —, 28
Successeur, 118
  état —, 162
  d'un naturel, 69
  ordinal —, 106
Suite, 62, 108
  ordinale, 108
  stationnaire, 273
Supremum, 272
Surjective
  fonction —, 58
Symétrie
  d'une relation, 59
Symbole
  extralogique, 63
Synchrone
  canal —, 215
  communication —, 215
Syntaxe, 141, 174
Système formel
  séquentiel, 160, 162
Système formel concurrent, 208

Systolique
algorithme —, 216

Tableau, 156
composante d'un —, 156
index d'un —, 156
Terminaison, 171
condition de —, 171
distribuée, 257
propriété de —, 171
Terminal
état —, 162
Théorie
des ensembles, 49
du premier ordre, 54, 63, 72, 75
ensembliste, 265
semi-formelle, 265
typée, 268
Thèse de Church, 196
Topologie, 282
d'Alexandrov, 283
de Scott, 284
Total
ordre —, 59
Totale
fonction —, 58
Transition, 162
exécutable, 162
Transitive
classe —, 101
Transitivité
d'une relation, 59
Treillis, 286
complet, 287
Tri
algorithme de —, 182
Triplet de Hoare, 145, 147, 177
Type, 268
d'un ensemble, 52
d'une expression, 143
d'une variable, 143, 162
restriction d'un —, 271

Unificateur principal, 135
Univers ensembliste, 114
Urelemente, 134

Valeur, 142
Variable, 142, 173
locale, 202
simple, 156
structurée, 156
typée, 268
type d'une —, 143, 162
Variant, 179
Vide
affectation —, 143
ensemble —, 53

Nouvelle Imprimerie Duculot S.A., Gembloux (Belgique). Tél. (081) 61 00 61

La première partie est consacrée à la théorie des ensembles. Le chapitre 1 donne une introduction philosophique et historique, les aspects techniques étant exposés au chapitre 2. La théorie des ensembles constitue à la fois une application essentielle de la logique du premier ordre, étudiée dans le premier volume, et la base théorique des méthodes formelles en programmation. La seconde partie est consacrée à ces méthodes et à leur illustration. Le chapitre 3 présente la logique de Hoare classique, permettant l'étude formelle des programmes séquentiels ; cette logique est adaptée à la programmation parallèle au chapitre 4. Le chapitre 5 examine la théorie du point fixe, qui permet l'étude systématique des programmes et langages de programmation de tous types, y compris des langages fonctionnels et logiques.
Cet ouvrage constitue donc une présentation claire et raisonnée des liens qui associent en profondeur *logique* et *étude des propriétés des programmes*.

Paul Gochet est professeur de logique et épistémologie à l'université de Liège. Il est membre de plusieurs sociétés savantes et a été titulaire de la chaire Francqui à l'université de Gand. On lui doit de nombreux travaux, notamment sur la philosophie de Quine.
Pascal Gribomont est chargé de cours en informatique et intelligence artificielle à l'université de Liège. Sa recherche concerne le développement, dans ces domaines, de méthodes formelles ainsi que leur application.

Éditions HERMES
14, rue Lantiez - 75017 Paris